Volume 16 Number 4

ACTA CYBERNETICA

Szeged, 2004

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and can be submitted by post or by email directly to the most competent Editor in the field covered by the paper (the Editor-in-Chief and the Managing Editor should be considered as an Editor). If submitting by email, use a printable form (ps or pdf) as an attachment to the email. The email addresses of the Editors can be found in the Acta Cybernetica home page. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the University of Szeged, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 2004 Numbers 3–4 of Volume 16 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, University of Szeged, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-546-396, Fax:(36)-(62)-546-397.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/kutatas/actacybernetica/.

W. Brauer
Institut für Informatik
Technische Universität München
D-80290 München
Germany

L. Budach
University of Postdam
Department of Computer Science
Am Neuen Palais 10
14415 Postdam, Germany

H. Bunke
Universität Bern
Institut für Informatik und
angewandte Mathematik
Längass strasse 51.
CH-3012 Bern, Switzerland

B. Courcelle
Universitè Bordeaux-1
LaBRI, 351 Cours de la Libèration
33405 TALENCE Cedex
France

J. Demetrovics
MTA SZTAKI
Budapest, Lágymányosi u. 11.
H-1111 Hungary

B. Dömölki
IQSOFT
Budapest, Teleki Blanka u. 15-17.
H-1142 Hungary

J. Engelfriet
Leiden University
LIACS
P.O. Box 9512, 2300 RA Leiden
The Netherlands

Z. Ésik
University of Szeged
Department of Foundations of
Computer Science
Szeged, Aradi vértanúk tere 1.
H-6720 Hungary

A. Kelemenová
Institute of Mathematics and
Computer Science
Silesian University at Opava
761 01 Opava, Czech Republic

L. Lovász
Eötvös Loránd University
Department of Computer Science
Budapest, Kecskeméti u. 10-12.
H-1053 Hungary

G. Păun
Institute of Mathematics
Romanian Academy
P.O.Box 1-764, RO-70700
Bucuresti, Romania

A. Prékopa
Eötvös Loránd University
Department of Operations Research
Budapest, Kecskeméti u. 10-12.
H-1053 Hungary

A. Salomaa
University of Turku
Department of Mathematics
SF-20500 Turku 50, Finland

L. Varga
Eötvös Loránd University
Department of General Computer Science
Budapest, Pázmány Péter sétány 1/c.
H-1117 Hungary

H. Vogler
Dresden University of Technology
Department of Computer Science
Foundations of Programming
D-01062 Dresden, Germany

G. Wöginger
Department of Matematics
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands

Preface

This issue of *Acta Cybernetica*, besides four regular papers, contains six papers whose preliminary versions appeared in Hungarian language in the proceedings of the 1st Conference on Hungarian Computational Linguistics, see `http://www.inf.u-szeged.hu/mszny2003/`. The conference was held in Szeged on December 10-11, 2003 and its aim was to provide a forum for researchers working on Hungarian computational linguistics and speech processing.

After the conference, the authors were invited to submit completed versions of their papers to *Acta Cybernetica*. All submitted papers were then subjected to the normal refereeing process of the journal. Altogether eight manuscripts were submitted, out of which six have been accepted.

We thank the authors and the referees for their help in the preparation of this issue.

The 2nd Conference on Hungarian Computational Linguistics will be held also in Szeged on December 9-10, 2004.

Zoltán Alexin
Tibor Gyimóthy
Guest Editors

Zoltán Fülöp
Managing Editor

Acta Cybernetica 16 (2004) 509–544.

A Pumping Lemma and Decidability Problems for Recognizable Tree Series*

Björn Borchardt†

Abstract

In the present paper we show that given a tree series S, which is accepted by (a) a deterministic bottom-up finite state weighted tree automaton (for short: bu-w-fta) or (b) a non-deterministic bu-w-fta over a locally finite semiring, there exists for every input tree $t \in \mathrm{supp}(S)$ a decomposition $t = C'[C[s]]$ into contexts C, C' and an input tree s as well as there exist semiring elements a, a', b, b', c such that the equation $(S, C'[C^n[s]]) = a' \odot a^n \odot c \odot b^n \odot b'$ holds for every non-negative integer n. In order to prove this pumping lemma we extend the power-set construction of classical theories and show that for every non-deterministic bu-w-fta over a locally finite semiring there exists an equivalent deterministic one. By applying the pumping lemma we prove the decidability of a tree series S being constant on its support, S being constant, S being boolean, the support of S being the empty set, and the support of S being a finite set provided that S is accepted by (a) a deterministic bu-w-fta over a commutative semiring or (b) a non-deterministic bu-w-fta over a locally finite commutative semiring.

1 Introduction

Finite state automata (for short: fsa) can be generalized in several ways: in [Sch61] fsa were enriched by weights (or: costs, multiplicities), which are taken from a semiring. This leads to the model of finite state weighted automata (fwa). The idea is that every run on an input string has a weight, which is obtained by multiplying the weights of the applied transitions. Also leaving the system is reflected in weights, which depends on the state where the run ends. Non-determinism is finally handled by summing up the weights of all runs multiplied with the appropriate final weight. Thus an fwa accepts every input string with a weight, which is a semiring element. A survey paper on the theory of fwa is [Kui97b] (also cf. [Eil74, KS86, BR88]), while in [Moh97, BGW00, DK03] recent results are presented.

*Research was financially supported by the German Research Council under grant (DFG, GRK 334/3).

†Dresden University of Technology, Faculty of Computer Science, D-01062 Dresden. E-mail: borchard@tcs.inf.tu-dresden.de

In [BR82] the generalization of adding weights to the transitions was applied to finite state tree automata (cf. [Eng75, GS84, GS97]), which yields the model of finite state weighted tree automata (or: concept of recognizable tree series, also cf. [Boz91, Boz99]). In this paper we use the notion of bottom-up finite state weighted tree automata (for short: bu-w-fta), which are tuples $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, where Q is a finite set (of states), Σ is a ranked alphabet (of input symbols), $\mathcal{A} = (A, \oplus, \odot, \mathbf{0}, \mathbf{1})$ is a semiring, $\nu : Q \longrightarrow A$ is a (final weight) mapping, and $\mu = (\mu_k \mid k \in \mathbb{N})$ is a family of mappings $\mu_k : \Sigma^{(k)} \longrightarrow A^{Q^k \times Q}$, in which the transitions and their weights are encoded. Similar to fwa every run of a bu-w-fta on an input tree t causes a weight, which is obtained by multiplying the weights of the applied transitions and finally multiplying this product with the appropriate final weight $\nu(q)$ assuming that the considered run ends in state q. Non-determinism is handled by summing up the weights of all runs. Thus a bu-w-fta accepts every tree with a weight, which is taken from the underlying semiring and hence its semantics is a tree series. Note that every state of a bu-w-fta is potentially a final state and thus every run is successful. We observe that the concepts of weighted grammars (cf. [AB87]), representable tree series (cf. [Boz94, Boz97]), and K-Σ-algebras (cf. [BA89, Boz99]) are strongly related (and equally powerful) to the above two concepts. We also note that besides the aforementioned concepts there exist more weighted tree automata models, e.g., $\mathcal{A}$-cost automata of [Sei94], $\mathcal{A}'$-tree automata of [Kui97a], and finite state weighted tree automata with final states of [BV03]. In Section 3 of this paper we compare the power of these models. A survey on recognizable tree series can be found in [ÉK03], while further results are presented in e.g., [Boz91, FSW94, Boz01, Bor03, DPV03]. We note that weighted tree automata are instances of tree series transducers, which recently were deeply investigated (cf. e.g., [Kui99, EFV02, FV03]).

Let us now answer the question why we introduce bu-w-fta rather than using one of the existing notions of recognizable tree series. In classical automata theory it is a common strategy to prove theorems by additionally assuming that the given device is deterministic and thereby using that deterministic and non-deterministic devices are equally powerful. We would like to prove results on recognizable tree series in the same way. This requires a notion of determinism, which, to the best of our knowledge, only exists for weighted tree automata of [BV03]. Section 4 of the aforementioned paper provides a determinization construction, which extends the power-set construction of classical theories by associating weights to the states. Lemma 6.1 of [BV03] states that the extended power-set construction yields an equivalent deterministic device provided that the underlying semiring is a locally finite *semifield* (which is a semiring with multiplicative inverses). Hence results which are proven for deterministic devices also hold for non-deterministic automata over locally finite semifields. By equipping finite state weighted tree automata of [BV03] with final weights and thereby considering bu-w-fta we can prove stronger results: similar to [BV03] we extend the power-set constructions of classical theories. As done in the aforementioned paper we associate weights to the states. By considering bu-w-fta rather than bottom-up finite state weighted tree automata

with final states the weights of the transitions of the constructed device can be defined such that all non-trivial computations of the automaton are shifted to the final weight mapping (cf. Definition 4.1). Thereby we obtain an equivalent deterministic bu-w-fta provided that the given bu-w-fta is defined over a locally finite *semiring* (cf. Theorem 4.8). Thus statements, which are proven for deterministic bu-w-fta, also hold for non-deterministic devices, if the underlying algebraic structure is a locally finite semiring.

We also prove a pumping lemma for recognizable tree series. In classical theories pumping lemmata state that, roughly speaking, parts of the input tree can be pumped such that recognizability is preserved. When considering bu-w-fta we would like to know how pumping is reflected in the weight the pumped tree is accepted with. Being more precise, in Theorem 5.6 we show that there exists a non-negative integer $m \in \mathbb{N}$ such that for every input tree $t \in \mathrm{supp}(S)$, which is contained in the support of S (i.e., t is mapped to a non-zero semiring element), and for every path of t of length $\geq m$ there exists a decomposition $t = C'[C[s]]$ along this path and semiring elements $a, a', b, b', c \in A$ such that $(S, C'[C^n[s]]) = a' \odot a^n \odot c \odot b^n \odot b'$ for every non-negative integer $n \in \mathbb{N}$. The pumping lemma assumes a deterministic bu-w-fta (or a non-deterministic device such that there exists an equivalent deterministic automaton). This is due to the pumping: in classical theories one can pump a context provided that there exists a run on this context, which starts and ends in the same state. There also might be additional runs, but they do not affect the accepting behavior. In weighted automata theory every run (with a non-zero weight) contributes to the weight an input tree is accepted with. Hence we restrict ourselves to deterministic devices and thereby apply the fact that in a deterministic device for every input tree there is at most one run (in our notion: there is at most one run with a non-zero weight). We note that in [BR82] a pumping lemma is proven for the concept of recognizable tree series. Theorem 9.2 of the aforementioned paper states that for every recognizable tree series S over a field there exists a constant m such that for every tree t of height $\geq m$, which is contained in the support of S, there exists a decomposition $t = C_1[C_2[C_3[\alpha]]]$ into contexts C_1, C_2, C_3 and a nullary input symbol α such that $C_1[C_2^*[C_3[\alpha]]] \cap \mathrm{supp}(S)$ is an infinite set. It is easily seen that Theorem 5.6 of the present paper generalizes the pumping lemma of [BR82] provided that the tree series is accepted by a deterministic device.

Similar to classical theories the pumping lemma can be applied for showing that a tree series is not accepted by a deterministic bu-w-fta. We prove that the particular tree series which maps every tree to its height is not recognized by a deterministic bu-w-fta over the arctic semiring. Since the set of all trees over some ranked alphabet is a recognizable tree language (i.e., a recognizable tree series over the Boolean semiring), we thereby show that recognizability is in general not preserved by associating weights to the transitions. The pumping lemma can also be used for deciding some common properties on tree series, e.g., is a given tree series constant on its support, constant, boolean, or is its support the empty or a finite set. We prove that all the aforementioned properties are decidable provided that the given tree series is accepted by (a) a deterministic bu-w-fta over

a commutative semiring or (b) a non-deterministic bu-w-fta over a locally finite, commutative semiring. The decidability result of a tree series having finite support additionally assumes a zero-divisor free semiring. We note that in [Boz91] ([Boz97]) it is shown under the assumption that the underlying algebraic structure is a field (the semiring $\mathbb{N}$ of all non-negative integers or the semiring $\mathbb{R}_+$ of all non-negative reals, respectively) that the equivalence problem, i.e., are two recognizable tree series equal, and the minimization problem, i.e., is an automaton which accepts a given recognizable tree series minimal, are decidable.

This paper is organized as follows: In Section 2 we recall well-known notions on trees, semirings, and formal tree series. The concept of bu-w-fta is introduced in Section 3, where we also compare bu-w-fta with existing models of recognizable tree series. We investigate the determinization of bu-w-fta in Section 4. In Section 5 we prove pumping lemmata, which we apply in Section 6, where we present several decidability results.

2 Preliminaries

2.1 Notions on Trees

The sets of all non-negative and positive integers are denoted by $\mathbb{N} = \{0, 1, \ldots\}$ and $\mathbb{N}_+ = \{1, 2, \ldots\}$, respectively. The *star* $\mathbb{N}^*$ *of* $\mathbb{N}$ is defined to be the set $\mathbb{N}^* = \bigcup_{i \in \mathbb{N}} \mathbb{N}^i$, where $\mathbb{N}^0 = \{\varepsilon\}$ and $\mathbb{N}^{i+1} = \{n.w \mid n \in \mathbb{N}, w \in \mathbb{N}^i\}$ for every non-negative integer $i \in \mathbb{N}$. We note that $v.w$ denotes the concatenation of $v, w \in \mathbb{N}^*$. Moreover, for every two non-negative integers $m, n \in \mathbb{N}$ let $[m, n]$ be the interval $\{m, m+1, \ldots, n\}$ provided that $m \leq n$. Otherwise we set $[m, n] = \emptyset$. As usual we write $[n]$ rather than $[1, n]$. If S is a set, then the *cardinality* and the *power set* of S are denoted by $\mathrm{card}(S)$ and $\mathfrak{P}(S)$, respectively. Now let Σ be a non-empty finite set and $\mathrm{rk} : \Sigma \longrightarrow \mathbb{N}$ be a mapping. The tuple (Σ, rk) is called *ranked alphabet.* Throughout this paper we will be short in notation and write Σ rather than (Σ, rk). For every non-negative integer $k \in \mathbb{N}$ we define the set $\Sigma^{(k)} = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = k\}$ of all symbols of Σ, which have rank k. An element $\sigma \in \Sigma^{(k)}$ is also written as $\sigma^{(k)}$.

Now let $n \in \mathbb{N}$ be a non-negative integer and $X_n = \{x_1, \ldots, x_n\}$ be a set of variables disjoint with Σ. The set $T_\Sigma(X_n)$ of *(finite, labeled, and ordered) trees over* Σ *(indexed by the set* X_n*)* is defined to be the smallest subset of $(\Sigma \cup X_n \cup \{(,)\} \cup \{,\})^*$ such that (i) $X_n \cup \Sigma^{(0)} \subseteq T_\Sigma(X_n)$ and (ii) $\sigma(t_1, \ldots, t_k) \in T_\Sigma(X_n)$ for every positive integer $k \in \mathbb{N}_+$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and input trees $t_1, \ldots, t_k \in T_\Sigma(X_n)$. The set $T_\Sigma(X_0)$ is denoted by T_Σ. The *substitution of* $x_1, \ldots, x_n$ *by* $s_1, \ldots, s_n \in T_\Sigma(X_n)$ *in* $t \in T_\Sigma(X_n)$ is the tree $t[s_1, \ldots, s_n] \in T_\Sigma(X_n)$ (as a shorthand for $t[x_1 \leftarrow s_1, \ldots, x_n \leftarrow s_n]$), where for every index $j \in [n]$ every occurrence of x_j in t is replaced by s_j. A tree $t \in T_\Sigma(X_n)$ is called Σ-*n-context* or *context*, if every variable $x \in X_n$ occurs precisely once in t. The set of all Σ-n-contexts is denoted by $C_\Sigma(X_n)$. The following observation shows that the set of Σ-1-contexts could also be defined by induction on its structure.

Observation 2.1. *Let $C \in T_\Sigma(X_1)$. It holds that $C \in C_\Sigma(X_1)$, if and only if C is the trivial context x_1 or $C = \sigma(t_1, \ldots, t_{i-1}, C', t_{i+1}, \ldots, t_k)$ for some non-negative integers $k \in \mathbb{N}$ and $i \in [k]$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, context $C' \in C_\Sigma(X_1)$, and trees $t_j \in T_\Sigma$ for every index $j \in [k] \setminus \{i\}$.*

Now let $t \in T_\Sigma(X_n)$ be a tree for some non-negative integer $n \in \mathbb{N}$. The *size* and *height of* t are inductively defined by $\mathrm{size}(x) = \mathrm{height}(x) = 1$ for every variable $x \in X_n$. Moreover, $\mathrm{size}(t) = 1 + \sum_{i \in [k]} \mathrm{size}(t_i)$ and $\mathrm{height}(t) = 1 + \max\{\mathrm{height}(t_i) \mid i \in [k]\}$ provided that $t = \sigma(t_1, \ldots, t_k) \in T_\Sigma(n)$ for some non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and trees $t_1, \ldots, t_k \in T_\Sigma(n)$. The set of *paths of* t is defined to be the image of the mapping $\mathrm{paths} : T_\Sigma(X_n) \longrightarrow \mathfrak{P}(\mathbb{N}^*)$, which is given by $\mathrm{paths}(t) = \{\varepsilon\} \cup \{i.w \mid i \in [k], w \in \mathrm{paths}(t_i)\}$. The *length of a path* $w = w_1 \ldots w_n \in \mathrm{paths}(t)$, where $w_i \in \mathbb{N}$ for every index $i \in [n]$, is defined to be $\mathrm{length}(w) = n$. We note that one could also look on the set $\mathrm{paths}(t)$ as the set of positions of t.

Observation 2.2. *Let $t \in T_\Sigma$ be a tree. The length of a longest path of t is* $\mathrm{height}(t) - 1$.

Let us finally define the subtrees of a tree $t \in T_\Sigma$ in terms of a function $\mathrm{paths}(t) \to T_\Sigma$: the *subtree* t/w *of* $t \in T_\Sigma$ *at the node* $w \in \mathrm{paths}(t)$ is defined inductively as follows: if $w = \varepsilon$ is the empty word, then $t/w = t$ and, if $w = i.w'$ for some integer $i \in [k]$ and word $w' \in \mathrm{paths}(t_i)$, then $t/w = t_i/w'$.

2.2 Semirings

In this section we briefly recall the concept of semirings, which is essential in weighted automata theory. For a more detailed presentation of semirings we refer the reader to [HW98]. Let A be a non-empty set, $\oplus$ and $\odot$ binary associative operations on A, and $\mathbf{0}, \mathbf{1}$ elements of A. As usual, $\odot$ is assumed to have a higher binding power than $\oplus$. The tuple $\mathcal{A} = (A, \oplus, \odot, \mathbf{0}, \mathbf{1})$ is called *semiring*, if (i) $\mathbf{0}$ and $\mathbf{1}$ are the neutral elements of $\oplus$ and $\odot$, respectively ($a \oplus \mathbf{0} = a = \mathbf{0} \oplus a$ and $a \odot \mathbf{1} = a = \mathbf{1} \odot a$), (ii) $\oplus$ is commutative ($a \oplus b = b \oplus a$), (iii) $\odot$ is left- and right-distributive over $\oplus$ ($a \odot (b \oplus c) = a \odot b \oplus a \odot c$ and $(a \oplus b) \odot c = a \odot c \oplus b \odot c$), and (iv) $\mathbf{0}$ is absorbing ($a \odot \mathbf{0} = \mathbf{0} = \mathbf{0} \odot a$).

For the rest of this paper let $\mathcal{A} = (A, \oplus, \odot, \mathbf{0}, \mathbf{1})$ be a semiring. As usual we lift the operations $\oplus$ and $\odot$ to sets $A_1, A_2 \subseteq A$ by defining $A_1 \oplus A_2 = \{a_1 \oplus a_2 \mid a_1 \in A_1, a_2 \in A_2\}$ and $A_1 \odot A_2 = \{a_1 \odot a_2 \mid a_1 \in A_1, a_2 \in A_2\}$. The semiring $\mathcal{A}$ is called *commutative*, if $\odot$ is commutative. We will shorten notation as follows: for every finite index set $I = \{i_1, \ldots, i_n\}$ for some non-negative integer $n \in \mathbb{N}$ and semiring elements $a_{i_1}, \ldots, a_{i_n} \in A$ let

$$\sum_{i \in I} a_i = \begin{cases} a_{i_1} \oplus \cdots \oplus a_{i_n} & , \text{if } I \neq \emptyset, \\ \mathbf{0} & , \text{otherwise.} \end{cases}$$

The semiring $\mathcal{A}$ is called *locally finite*, if, for every finite subset A' of A, the closure $\langle A' \rangle_{\{\oplus, \odot\}}$ of A' under the semiring operations $\oplus$ and $\odot$ is again a finite set. Clearly,

every finite semiring is locally finite. Moreover, the min-max-semiring, which is defined below, is a locally finite semiring with an infinite carrier set.

Let us now present some well known semirings.

- The semiring of non-negative integers $Nat = (\mathbb{N}, +, \cdot, 0, 1)$ with the usual addition and multiplication. Nat can be used in automata theory for counting successful paths.
- The *Boolean* semiring $Bool = (\{0, 1\}, \vee, \wedge, 0, 1)$ with disjunction and conjunction. This semiring has a highly theoretical meaning, since there is a one-to-one correspondence between weighted (tree) automata over the Boolean semiring and unweighted (tree) automata.
- The *Tropical semiring* $Trop = (\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$, in which the semiring addition and multiplication are the natural extension of the minimum operation and addition of the non-negative integers to $\mathbb{N} \cup \{+\infty\}$, respectively. $Trop$ can be used for calculating shortest paths or minimal costs.
- The *Arctic semiring* $Arct = (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, where, similar to the Tropical semiring, the semiring addition and multiplication are the extension of the maximum operation and addition of the non-negative integers to $\mathbb{N} \cup \{-\infty\}$, respectively. $Arct$ is used for calculating longest paths or critical costs.
- The min-max-*semiring* $MinMax = (\mathbb{N} \cup \{\pm\infty\}, \min, \max, +\infty, -\infty)$, in which the semiring addition and multiplication are the natural extension of the minimum and maximum operations of the non-negative integers to $\mathbb{N} \cup \{\pm\infty\}$, respectively. $MinMax$ can be used for solving capacity problems.

We note that many number structures, e.g., the integers $Int = (\mathbb{Z}, +, \cdot, 0, 1)$, the rational numbers $Rat = (\mathbb{Q}, +, \cdot, 0, 1)$, the real numbers $Real = (\mathbb{R}, +, \cdot, 0, 1)$, and the complex numbers $Comp = (\mathbb{C}, +, \cdot, 0, 1)$ with the usual addition and multiplication are semirings.

2.3 Formal Tree Series

Let us now recall the concept of formal tree series. A *(formal) tree series (over a ranked alphabet Σ and semiring $\mathcal{A}$)* is a mapping $S : T_\Sigma \longrightarrow A$. In what follows, we use another notation: the image $S(t) \in A$ of a tree $t \in T_\Sigma$ is called *coefficient of* t and, according to power series, which are known from analysis, the coefficient of t is denoted by (S, t). The tree series S now can be written as the sum $\sum_{t \in T_\Sigma} (S, t)\, t$. The set of all tree series over Σ and $\mathcal{A}$ is denoted by $\mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. The *support of* S is the set $\mathrm{supp}(S) = \{t \in T_\Sigma \mid (S, t) \neq \mathbf{0}\}$. A tree series S is called *boolean*, if $(S, t) \in \{\mathbf{0}, \mathbf{1}\}$ for every tree $t \in T_\Sigma$. Moreover, S is called *constant on its support*, if there exists a semiring element $a \in A$ such that $(S, t) = a$ for every tree $t \in \mathrm{supp}(S)$. A tree series S, which is constant on its support, is called *constant*

tree series, denoted by $S = \widetilde{a}$, if there exists a semiring element $a \in A$ such that $(S,t) = a$ for every tree $t \in T_\Sigma$.

We conclude this section by defining two operations on tree series. Let $\mathcal{A}$ be a semiring and $S, T \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ tree series. The *sum* $S \oplus T$, and the *Hadamard product* $S \odot T$ are defined for every tree $t \in T_\Sigma$ by $(S \oplus T) = (S,t) \oplus (T,t)$ and $(S \odot T, t) = (S,t) \odot (T,t)$, respectively.

For more details on formal tree series we refer the reader to [Kui99].

3 Bottom-Up Finite State Weighted Tree Automata

In this section we introduce bottom-up finite state weighted tree automata with final weights. There is a tight relationship between bottom-up finite state weighted tree automata of [BV03] which have final states rather than final weights, and the devices, which we define below. This relationship will be discussed in the course of this section as well as the relationships to further weighted tree automata models. We also present an application, namely tree pattern matching. We conclude this section by proving that the cross product of two bottom-up finite state weighted tree automata M_1 and M_2 accepts the Hadamard product of the tree series, which are accepted by M_1 and M_2.

Let us start this section by defining tree representations. Tree representations encode the transitions and their weights. We note that for technical reasons at this time we do *not* assume a finite set of states.

Definition 3.1 (Tree representation, cf. [BV03], Definition 3.1). *Let Q be a not necessarily finite set (of states), Σ a ranked alphabet (of input symbols), and $\mathcal{A}$ a semiring. A* (bottom-up) tree representation (over Q, Σ, and $\mathcal{A}$) *is a family $\mu = (\mu_k \mid k \in \mathbb{N})$ of mappings $\mu_k : \Sigma^{(k)} \longrightarrow A^{Q^k \times Q}$. A tree representation is called* finite, *if the underlying set Q of states is finite. Moreover, a tree representation μ is called* deterministic, *if for every non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and k-tuple of states $(q_1, \ldots, q_k) \in Q^k$ there is at most one state $q \in Q$ such that $\mu_k(\sigma)_{(q_1,\ldots,q_k),q} \neq \mathbf{0}$.*

Every finite tree representation μ induces a family of mappings $(\overline{\mu_k(\sigma)} \mid k \in \mathbb{N}, \sigma \in \Sigma^{(k)})$ in the following way:

$$\begin{aligned} &\overline{\mu_k(\sigma)} : A^Q \times \ldots \times A^Q \longrightarrow A^Q : \\ &\overline{\mu_k(\sigma)}(V_1, \ldots, V_k)_q = \sum_{(q_1,\ldots,q_k) \in Q^k} (V_1)_{q_1} \odot \ldots \odot (V_k)_{q_k} \odot \mu_k(\sigma)_{(q_1,\ldots,q_k),q} \end{aligned}$$

for every state $q \in Q$ and vectors $V_1, \ldots, V_k \in A^Q$. We observe that $\left(A^Q, (\overline{\mu_k(\sigma)} \mid k \in \mathbb{N}, \sigma \in \Sigma^{(k)})\right)$ is a Σ-algebra. Its unique homomorphism $h_\mu : T_\Sigma \to A^Q$ is given for every non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and trees $t_1, \ldots t_k \in T_\Sigma$ by

$$h_\mu(\sigma(t_1, \ldots t_k)) = \overline{\mu_k(\sigma)}(h_\mu(t_1), \ldots, h_\mu(t_k)).$$

We call $h_\mu(t)$ the *characteristic vector of the tree $t \in T_\Sigma$ (with respect to the tree representation μ).* Let us now define bottom-up weighted tree automata. For technical reasons we define automata with an infinite set of states as well as automata with a finite set of states.

Definition 3.2 (Bottom-up (finite state) tree automata). *Let Q be a set (of states), Σ a ranked alphabet (of input symbols), $\mathcal{A}$ a semiring, $\nu : Q \longrightarrow A$ a mapping (final weight mapping), and μ a tree representation. The tuple $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ is called* bottom-up weighted tree automaton (with final weight mapping, for short bu-w-ta). *A bu-w-ta is called* deterministic, *if its tree representation is deterministic. A bu-w-ta is called* bottom-up finite state weighted tree automaton (for short bu-w-fta), *if its tree representation is finite. The tree series S_M, which is* accepted or recognized *by a (finite) bu-w-fta M, is defined for every tree $t \in T_\Sigma$ by $(S_M, t) = \sum_{q \in Q} h_\mu(t)_q \odot \nu(q)$. We denote by $\mathcal{A}^{n,bu}\langle\langle T_\Sigma \rangle\rangle$ and $\mathcal{A}^{d,bu}\langle\langle T_\Sigma \rangle\rangle$ the classes of all tree series, which are accepted by bu-w-fta and deterministic bu-w-fta, respectively.*

In Example 3.3 we present a bu-w-fta over the Arctic semiring, which accepts every input tree with its height. Note that in Example 5.9 we prove that this tree series is not accepted by any deterministic bu-w-fta over the Arctic semiring, which shows that deterministic and non-deterministic bu-w-fta are in general not equally powerful.

Example 3.3 ($\sum_{t \in T_\Sigma}$ height(t) t accepted by some non-deterministic bu-w-fta). Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ the bu-w-fta, which is defined by $Q = \{q, q_0\}$, $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$, $\nu(q) = 0$, $\nu(q_o) = -\infty$, $\mathcal{A} = Arct$,

$$\mu_0(\alpha)_{(),q} = 1, \quad \mu_0(\alpha)_{(),q_0} = 0,$$
$$\mu_2(\sigma)_{(q,q_0),q} = 1, \quad \mu_2(\sigma)_{(q_0,q),q} = 1, \quad \mu_2(\sigma)_{(q_0,q_0),q_0} = 0,$$

and, for every three states $q_1, q_2, q_3 \in Q$, for which $\mu_2(\sigma)_{(q_1,q_2),q_3}$ is not yet defined, let $\mu_2(\sigma)_{(q_1,q_2),q_3} = -\infty$. The following straightforward inductive proof shows that $h_\mu(t)_q = \text{height}(t)$ and $h_\mu(t)_{q_0} = 0$ for every tree $t \in T_\Sigma$: if $t = \alpha$, then

$$h_\mu(t)_q = \overline{\mu_0(\alpha)}()_q = \mu_0(\alpha)_{(),q} = 1 = \text{height}(t)$$

and similarly $h_\mu(t)_{q_0} = 0$. If $t = \sigma(t_1, t_2)$ for some trees $t_1, t_2 \in T_\Sigma$, then

$$\begin{aligned}
&h_\mu(\sigma(t_1, t_2))_q \\
&= \overline{\mu_2(\sigma)}(h_\mu(t_1), h_\mu(t_2))_q \\
&= \max\{h_\mu(t_1)_{p_1} + h_\mu(t_2)_{p_2} + \mu_2(\sigma)_{(p_1,p_2),q} \mid (p_1, p_2) \in Q^2\} \\
&= \max\{h_\mu(t_1)_q + h_\mu(t_2)_{q_0} + \mu_2(\sigma)_{(q,q_0),q}, h_\mu(t_1)_{q_0} + h_\mu(t_2)_q + \mu_2(\sigma)_{(q_0,q),q}, -\infty\} \\
&= \max\{\text{height}(t_1) + 0 + 1, 0 + \text{height}(t_2) + 1, -\infty\} \\
&= 1 + \max\{\text{height}(t_1), \text{height}(t_2)\} \\
&= \text{height}(t).
\end{aligned}$$

By a similar calculation one easily proves that $h_\mu(\sigma(t_1,t_2))_{q_0} = 0$. Hence M accepts every input tree $t \in T_\Sigma$ with $(S_M,t) = \text{height}(t)$.

Let us now discuss the relationship between (finite) bu-w-fta and other weighted tree automata models. Obviously representable tree series, which are considered in e.g. [Boz94] are precisely the tree series, which are accepted by non-deterministic bu-w-fta. Let us now compare bu-w-fta with bottom-up finite state weighted tree automata (with final states), which were introduced in [BV03]. The latter devices are defined to be tuples $M = (Q,\Sigma,Q_d,\mathcal{A},\mu)$, where Q, Σ, $\mathcal{A}$, and μ are as in Definition 3.2 and Q_d is a subset of Q (of final states). M accepts every input tree $t \in T_\Sigma$ with the weight $(S_M,t) = \sum_{q\in Q_d} h_\mu(t)_q$. A bottom-up finite state weighted tree automaton M with final states can be modeled by a bu-w-fta by taking the same set of states, ranked alphabet, semiring, and tree representation. The final weight mapping maps every final state to $\mathbf{1}$ and every non-final state to $\mathbf{0}$. The equivalence of both devices is easily seen. Conversely, a bu-w-fta $M = (Q,\Sigma,\nu,\mathcal{A},\mu)$ with final weight mapping can be modeled by a bottom-up finite state weighted tree automaton $M' = (Q',\Sigma,Q'_d,\mathcal{A},\mu')$ with final states by introducing a new state $* \notin Q$, which is the unique final state: we set $Q' = Q \cup \{*\}$, $Q'_d = \{*\}$, and for every non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and states $q'_1,\ldots,q'_k,q' \in Q'$,

$$\mu'_k(\sigma)_{(q'_1,\ldots,q'_k),q'} = \begin{cases} \mu_k(\sigma)_{(q'_1,\ldots,q'_k),q'} & \text{, if } q'_1,\ldots,q'_k,q' \in Q, \\ \sum_{q\in Q} \mu_k(\sigma)_{(q'_1,\ldots,q'_k),q} \odot \nu(q) & \text{, if } q'_1,\ldots,q'_k \in Q \text{ and } q' = *, \\ \mathbf{0} & \text{, otherwise.} \end{cases}$$

The (inductive) proof of equivalence is very straightforward. We therefore leave it to the reader. By the above two constructions it is shown that the two non-deterministic models of bottom-up finite state weighted tree automata are equally powerful. Unfortunately, the latter construction does not preserve determinism. Being more precise, deterministic bu-w-fta are in general more powerful than deterministic bottom-up finite state weighted tree automata with final states.

In Section 3 of [BV03] it is shown that bottom-up finite state weighted tree automata with final states and hence bu-w-fta of the present paper are particular $\mathcal{A}$-cost automata of [Sei94] and $\mathcal{A}'$-tree automata of [Kui97a] (by considering the equally powerful top-down devices).

Let us now compare bu-w-fta with the concept of recognizable tree series, which was introduced in [BR82] (also cf. [Boz91, Boz99, ÉK03]). For the algebraic notions we refer the reader to any good algebra textbook. A recognizable tree series is defined in terms of a Σ-algebra $\mathcal{V} = (V,a)$, where V is a vector space and $a = (a_\sigma : V^k \longrightarrow V \mid \sigma \in \Sigma^{(k)})$ is a family of multi-linear mappings. As usual, the family a of multi-linear mappings is extended to a mapping $\mu_\mathcal{V} : T_\Sigma \longrightarrow V$, which is inductively defined for every non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and trees $t_1,\ldots,t_k \in T_\Sigma$ by $\mu_\mathcal{V}(\sigma(t_1,\ldots,t_k)) = a_\sigma(\mu_\mathcal{V}(t_1),\ldots,\mu_\mathcal{V}(t_k))$. A tree series $S \in \mathcal{A}\langle\langle T_\Sigma\rangle\rangle$ is recognizable, if there exists a realization $(\mathcal{V},\varphi)$ of S, which is a pair consisting of a Σ-algebra $\mathcal{V}$ over a finite dimensional vector space

(as introduced above) and a linear form $\varphi : V \longrightarrow A$ such that $S = \varphi(\mu_{\mathcal{V}})$. We now briefly show that a tree series is recognizable in the sense of [BR82], if and only if it is accepted by a bu-w-fta provided that the underlying semiring is commutative. First let a bu-w-fta $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ be given. A realization $(\mathcal{V}, \varphi)$ of the tree series S_M, which is accepted by M, can be defined as follows: the underlying Σ-algebra $\mathcal{V} = (V, a)$ is given by the vector space $V = A^Q$ and the sequence $a = (a_\sigma : V^k \longrightarrow V \mid k \in \mathbb{N}, \sigma \in \Sigma^{(k)})$ of multi-linear mappings $a_\sigma = \overline{\mu_k(\sigma)}$ for every k-ary input symbol $\sigma \in \Sigma^{(k)}$. We observe that $\overline{\mu_k(\sigma)}$ is a multi-linear mapping provided that the underlying semiring is commutative. We define the linear form φ for every vector $v \in V$ by $\varphi(v) = \sum_{q \in Q} v_q \odot \nu(q)$. The (inductive) proof of correctness is very straightforward and hence left to the reader. Conversely, let S be a recognizable tree series in the sense of [BR82], i.e., there exists a finite dimensional realization $(\mathcal{V}, \varphi)$ with $\mathcal{V} = (V, a)$ and $a = (a_\sigma : V^k \longrightarrow V \mid k \in \mathbb{N}, \sigma \in \Sigma^{(k)})$ of S. We define a bu-w-fta $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, which accepts the tree series S, as follows: Q is a basis of the vector space V. Moreover, for all states $q_1, \ldots, q_k, q \in Q$ we define the final weight mapping ν and the tree representation μ by $\nu(q) = \varphi(q)$ and $\mu_k(\sigma)_{(q_1,\ldots,q_k),q} = a_\sigma(q_1, \ldots, q_k)_q$, respectively. One easily proves by induction on the structure of the input tree $t \in T_\Sigma$ that the bu-w-fta M accepts the recognizable tree series (in the sense of [BR82]). Summing up, we have shown that our notion of recognizable tree series coincides with the classical notion of [BR82] provided that the underlying semiring is commutative.

Let us now present an application of weighted tree automata.

Example 3.4 (Tree pattern matching, also cf. [FSW94]). Consider a tree $t \in T_\Sigma$ and a pattern C. We would like to find all occurrences of C in t and, roughly speaking, give references to the root of the occurrences of C. This can be formalized as follows: let $t \in T_\Sigma$ and $C \in C_\Sigma(X_m)$ for some non-negative integer $m \in \mathbb{N}$. We call C *pattern of* t *at* $w \in \text{paths}(t)$, if $t/w = C[t_1, \ldots, t_m]$ for some trees $t_1, \ldots, t_m \in T_\Sigma$. Before we define the tree series S_C, which maps every tree t to the set of all $w \in \text{paths}(t)$ such that C is a pattern of t at w, we introduce the semiring, which we will work with. Consider the tuple $(\mathfrak{P}(\mathbb{N}^*), \cup, \circ, \emptyset, \{\varepsilon\})$, where $\circ$ is the binary operation on $\mathfrak{P}(\mathbb{N}^*)$, which is defined for every two subsets $A, B \in \mathfrak{P}(\mathbb{N}^*)$ by $A \circ B = \{b.a \in \mathbb{N}^* \mid a \in A, b \in B\}$. Recall that $b.a$ denotes the concatenation of the words b and a. One easily proves that $(\mathfrak{P}(\mathbb{N}^*), \cup, \circ, \emptyset, \{\varepsilon\})$ is a semiring.

Let us now define the tree series S_C over the semiring $(\mathfrak{P}(\mathbb{N}^*), \cup, \circ, \emptyset, \{\varepsilon\})$ as follows: for every tree $t \in T_\Sigma$ let

$$(S_C, t) = \{w \in \mathbb{N}^* \mid (\exists t_1, \ldots, t_m \in T_\Sigma) : t/w = C[t_1, \ldots, t_m]\}.$$

We claim that the tree series S_C can be computed by a bu-w-fta over the semiring $\mathfrak{P}(\mathbb{N}^*)$. Since the general case is very technical and the intention of this paper is to prove a pumping lemma rather than discussing tree pattern matching, we now restrict ourselves to the particular ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$ and pattern $C = \sigma(\sigma(\alpha, \alpha), x_1)$, and note that the general case is very similar. Let us now define a bu-w-fta $M_C = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, which accepts the tree series S_C. M_C is given by

$Q = \{q_\perp, q_\alpha, q_{\sigma(\alpha,\alpha)}, q_C\}$, $\nu(q_C) = 1$ and $\nu(q) = 0$ for every state $q \in Q \setminus \{q_C\}$, $\mathcal{A} = (\mathfrak{P}(\mathbb{N}^*), \cup, \circ, \emptyset, \{\varepsilon\})$, and

$$\begin{aligned} \mu_0(\alpha)_{(),q_\perp} &= \{\varepsilon\}, & \mu_0(\alpha)_{(),q_\alpha} &= \{\varepsilon\}, \\ \mu_2(\sigma)_{(q_\perp,q_\perp),q_\perp} &= \{\varepsilon\}, & \mu_2(\sigma)_{(q_\alpha,q_\alpha),q_{\sigma(\alpha,\alpha)}} &= \{\varepsilon\}, \\ \mu_2(\sigma)_{(q_{\sigma(\alpha,\alpha)},q_\perp),q_C} &= \{\varepsilon\}. \end{aligned}$$

Moreover, for every state $q \in Q \setminus \{q_C\}$,

$$\mu_2(\sigma)_{(q_C,q),q_C} = \{1\}, \qquad \mu_2(\sigma)_{(q,q_C),q_C} = \{2\}.$$

Otherwise we set $\mu_2(\sigma)_{(q_1,q_2),q} = \emptyset$ for every three states $q_1, q_2, q \in Q$. Let us now briefly discuss the intended meaning of the states. This requires us to consider "runs" on an input tree t. If a "run" ends up in the state q_α, ($q_{\sigma(\alpha,\alpha)}$, $q_\perp$, respectively), then it has either weight $\emptyset$ or $\{\varepsilon\}$ and we have just met an α-tree (a $\sigma(\alpha,\alpha)$-tree, an arbitrary tree, respectively). If a "run" ends up in the state q_C, then again, either it has weight $\emptyset$ or we have met the pattern C while traversing the input tree and the weight of the "run" is $\{w\}$, where $w \in \mathrm{paths}(t)$ and $t/w = C[t']$ for some tree $t' \in T_\Sigma$. The inductive proof of correctness is very straightforward. We leave it to the reader.

Later on, for a given input tree $t \in T_\Sigma$ and bu-w-fta $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, we will work with the set $\widetilde{\mu}(t)$ of all those states $q \in Q$ such that, roughly speaking, there exists a "run" of the automaton M on t ending in state q such that every "transition associated to this run" has a weight different from zero. Formally, the mapping $\widetilde{\mu} : T_\Sigma \longrightarrow \mathfrak{P}(Q)$ is inductively defined for every input tree $t = \sigma(t_1, \ldots, t_k)$, where $k \in \mathbb{N}$ is a non-negative integer, $\sigma \in \Sigma^{(k)}$ is a k-ary input symbol, and $t_1, \ldots, t_k \in T_\Sigma$ are trees, by

$$\widetilde{\mu}(t) = \{q \in Q \mid (\forall i \in [k]), (\exists q_i \in \widetilde{\mu}(t_i)), \mu_k(\sigma)_{(q_1,\ldots,q_k),q} \neq \mathbf{0}\}.$$

Observation 3.5. *Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ be a bu-w-fta, $s, t \in T_\Sigma$ trees, and $C = \sigma(t_1, \ldots, t_{i-1}, x_1, t_{i+1}, \ldots, t_k)$ a context for some positive integers $k \in \mathbb{N}_+$ and $i \in [k]$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and trees $t_j \in T_\Sigma$ for every index $j \in [k] \setminus \{i\}$.*

(i) If $q \in Q \setminus \widetilde{\mu}(s)$, then $h_\mu(s)_q = \mathbf{0}$.

(ii) If $s \in \mathrm{supp}(S_M)$, then $\widetilde{\mu}(s) \neq \emptyset$.

(iii) If $\widetilde{\mu}(s) = \emptyset$, then $\widetilde{\mu}(C[s]) = \emptyset$.

(iv) If $\widetilde{\mu}(s) = \widetilde{\mu}(t)$, then $\widetilde{\mu}(C[s]) = \widetilde{\mu}(C[t])$.

(v) If M is a deterministic bu-w-fta, then $\widetilde{\mu}(s)$ is either the empty set or a singleton. In the latter case we identify $\widetilde{\mu}(s)$ with the state contained in $\widetilde{\mu}(s)$.

(vi) If M is a deterministic bu-w-fta and $\widetilde{\mu}(s) \in Q$, then $(S_M, s) = h_\mu(s)_{\widetilde{\mu}(s)} \odot \nu(\widetilde{\mu}(s))$.

By a repeated application of Observation 3.5 (iii) and (iv) we obtain the following statement:

Corollary 3.6. *Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ be a bu-w-fta, $s, t \in T_\Sigma$ trees, and $C \in C_\Sigma(X_1)$ a context.*

(i) If $\widetilde{\mu}(s) = \mathbb{0}$, then $\widetilde{\mu}(C[s]) = \mathbb{0}$.

(ii) If $\widetilde{\mu}(s) = \widetilde{\mu}(t)$, then $\widetilde{\mu}(C[s]) = \widetilde{\mu}(C[t])$.

In classical automata theory the cross product $\mathfrak{A}_1 \times \mathfrak{A}_2$ of two automata $\mathfrak{A}_1$ and $\mathfrak{A}_2$ is defined by setting the set of states (initial states, final states, respectively) of $\mathfrak{A}_1 \times \mathfrak{A}_2$ to the cross product of the sets of states (initial states, final states, respectively) of $\mathfrak{A}_1$ and $\mathfrak{A}_2$. The transitions are defined in the obvious way. It is well known that $\mathfrak{A}_1 \times \mathfrak{A}_2$ accepts the intersection of the languages, which are accepted by $\mathfrak{A}_1$ and $\mathfrak{A}_2$. We now define the cross product $M_1 \times M_2$ of bu-w-fta M_1 and M_2 and prove that, if the underlying semiring is commutative, then $M_1 \times M_2$ accepts the Hadamard product of the tree series, which are recognized by M_1 and M_2.

Definition 3.7 (Cross product). *Let $M_1 = (Q_1, \Sigma, \nu_1, \mathcal{A}, \mu_1)$ and $M_2 = (Q_2, \Sigma, \nu_2, \mathcal{A}, \mu_2)$ be bu-w-fta. The* cross product *of M_1 and M_2 is defined to be the bu-w-fta $M_1 \times M_2 = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, where $Q = Q_1 \times Q_2$, $\nu((p,q)) = \nu_1(p) \odot \nu_2(q)$ for every two states $p \in Q_1$ and $q \in Q_2$, and μ is defined for every non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and states $p_1, \ldots, p_k, p \in Q_1$, $q_1, \ldots, q_k, q \in Q_2$ by*

$$\mu_k(\sigma)_{((p_1,q_1),\ldots,(p_k,q_k)),(p,q)} = (\mu_1)_k(\sigma)_{(p_1,\ldots,p_k),p} \odot (\mu_2)_k(\sigma)_{(q_1,\ldots,q_k),q}.$$

Lemma 3.8. *Let $\mathcal{A}$ be a commutative semiring and M_1, M_2, and $M_1 \times M_2$ bu-w-fta as required/defined in Definition 3.7. It holds that $(S_{M_1 \times M_2}, t) = (S_{M_1}, t) \odot (S_{M_2}, t)$ for every input tree $t \in T_\Sigma$.*

Proof. Let us first show that the equation $h_\mu(t)_{(p,q)} = h_{\mu_1}(t)_p \odot h_{\mu_2}(t)_q$ holds for every two states $p \in Q_1$ and $q \in Q_2$, which we prove by induction on the structure of the input tree $t \in T_\Sigma$. Note that the induction base is covered by the induction step. Let $t = \sigma(t_1, \ldots, t_k)$ for some non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and trees $t_1, \ldots, t_k \in T_\Sigma$. For every two states $p \in Q_1$ and $q \in Q_2$,

$$\begin{aligned}
&h_\mu(\sigma(t_1, \ldots, t_k))_{(p,q)} \\
&= \sum_{\substack{((p_1,q_1),\ldots,(p_k,q_k)) \\ \in (Q_1 \times Q_2)^k}} h_\mu(t_1)_{(p_1,q_1)} \odot \cdots \odot h_\mu(t_k)_{(p_k,q_k)} \odot \mu_k(\sigma)_{((p_1,q_1),\ldots,(p_k,q_k)),(p,q)} \\
&= \sum_{\substack{((p_1,q_1),\ldots,(p_k,q_k)) \\ \in (Q_1 \times Q_2)^k}} (h_{\mu_1}(t_1)_{p_1} \odot h_{\mu_2}(t_1)_{q_1}) \odot \cdots \odot (h_{\mu_1}(t_k)_{p_k} \odot h_{\mu_2}(t_k)_{q_k}) \odot \\
&\qquad \odot \mu_k(\sigma)_{((p_1,q_1),\ldots,(p_k,q_k)),(p,q)} \\
&\qquad \text{(by induction hypothesis)}
\end{aligned}$$

$$= \sum_{\substack{((p_1,q_1),\ldots,(p_k,q_k)) \\ \in (Q_1\times Q_2)^k}} (h_{\mu_1}(t_1)_{p_1} \odot h_{\mu_2}(t_1)_{q_1}) \odot \cdots \odot (h_{\mu_1}(t_k)_{p_k} \odot h_{\mu_2}(t_k)_{q_k}) \odot$$

$$\odot (\mu_1)_k(\sigma)_{(p_1,\ldots,p_k),p} \odot (\mu_2)_k(\sigma)_{(q_1,\ldots,q_k),q}$$

$$= \sum_{\substack{(p_1,\ldots,p_k)\in Q_1^k \\ (q_1,\ldots,q_k)\in Q_2^k}} h_{\mu_1}(t_1)_{p_1} \odot \cdots \odot h_{\mu_1}(t_k)_{p_k} \odot (\mu_1)_k(\sigma)_{(p_1,\ldots,p_k),p} \odot$$

$$\odot h_{\mu_2}(t_1)_{q_1} \odot \cdots \odot h_{\mu_2}(t_k)_{q_k} \odot (\mu_2)_k(\sigma)_{(q_1,\ldots,q_k),q}$$

$$= \left(\sum_{(p_1,\ldots,p_k)\in Q_1^k} h_{\mu_1}(t_1)_{p_1} \odot \cdots \odot h_{\mu_1}(t_k)_{p_k} \odot (\mu_1)_k(\sigma)_{(p_1,\ldots,p_k),p} \right) \odot$$

$$\odot \left(\sum_{(q_1,\ldots,q_k)\in Q_2^k} h_{\mu_2}(t_1)_{q_1} \odot \cdots \odot h_{\mu_2}(t_k)_{q_k} \odot (\mu_2)_k(\sigma)_{(q_1,\ldots,q_k),q} \right)$$

$$= h_{\mu_1}(\sigma(t_1,\ldots,t_k))_p \odot h_{\mu_2}(\sigma(t_1,\ldots,t_k))_q.$$

This proves the claim. Let us now show that the lemma holds. We have

$$\begin{aligned}(S_{M_1\times M_2}) &= \sum_{(p,q)\in Q_1\times Q_2} h_\mu(t)_{(p,q)} \odot \nu((p,q)) \\ &= \sum_{(p,q)\in Q_1\times Q_2} h_{\mu_1}(t)_p \odot h_{\mu_2}(t)_q \odot \nu_1(p) \odot \nu_2(q) \\ &= \left(\sum_{p\in Q_1} h_{\mu_1}(t)_p \odot \nu_1(p) \right) \odot \left(\sum_{q\in Q_2} h_{\mu_2}(t)_q \odot \nu_2(q) \right) \\ &= (S_{M_1},t) \odot (S_{M_2},t),\end{aligned}$$

which proves the lemma. □

Corollary 3.9 (cf. Proposition 5.1 of [BR82]). *Let $\mathcal{A}$ be a commutative semiring.*

(i) If $S_1, S_2 \in \mathcal{A}^{n,bu}\langle\langle T_\Sigma\rangle\rangle$, then $S_1 \odot S_2 \in \mathcal{A}^{n,bu}\langle\langle T_\Sigma\rangle\rangle$.

(ii) If $S_1, S_2 \in \mathcal{A}^{d,bu}\langle\langle T_\Sigma\rangle\rangle$, then $S_1 \odot S_2 \in \mathcal{A}^{d,bu}\langle\langle T_\Sigma\rangle\rangle$.

Proof. The claims follow from Lemma 3.8 and the observation that the cross product $M_1 \times M_2$ of two deterministic bu-w-fta M_1 and M_2 again is a deterministic bu-w-fta. □

We note that Definition 3.7, Lemma 3.8, and Corollary 3.9 cover the corresponding theory of fwa (cf. [KS86]).

4 Determinization

In this section we construct for a given bu-w-fta M an equivalent deterministic device. To do so we introduce a natural extension of the power set construction known from the theory of bottom-up finite tree automata (cf. [GS84]). We encode the characteristic vector of a tree into the states of the generated automaton $det^p(M)$. It turns out that the extended power set construction is partial in the sense that there exist a bu-w-fta M and an equivalent deterministic bu-w-fta M' such that $det^p(M)$ is an infinite device.

For the rest of this section let

- $M = (Q, \Sigma, \nu, A, \mu)$ be a bu-w-fta,
- $q'_t = \bigcup_{q \in Q}\{(q, h_\mu(t)_q)\} \in \mathfrak{P}(Q \times A)$ be the state associated with the tree $t \in T_\Sigma$, and
- $Q' = \{q'_t \mid t \in T_\Sigma\}$ the set of all states, which are associated with a tree over Σ.

We observe that two states q'_t and $q'_{t'}$ are equal, if and only if the characteristic vectors $h_\mu(t)$ and $h_\mu(t')$ are equal. Later on we need to express the weight of a state $q \in Q$ of the given device M in a state $q' \in Q'$ of the constructed device $det^p(M)$. Therefore we define for every state $q' \in Q'$ of the constructed device the mapping $a_{q'} : Q \longrightarrow A$, which is defined for every state $q \in Q$ of the given device by

$$a_{q'}(q) = h_\mu(t)_q \qquad \text{assuming that} \qquad q' = q'_t \text{ for some } t \in T_\Sigma.$$

Hence, $q'_t = \bigcup_{q \in Q}\{(q, a_{q'_t}(q))\}$. Now let us introduce the extended power set construction. We note that the constructed automaton might have an infinite set of states, i.e., we only generate a bu-w-ta. Later on we show that, if the underlying semiring is locally finite, then M is a finite automaton (cf. Lemma 4.7).

Definition 4.1 (Extended power set construction). *Let $M = (Q, \Sigma, \nu, A, \mu)$ be a bu-w-fta. The bu-w-ta $det^p(M) = (Q', \Sigma, \nu', A, \mu')$ is defined for every non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and states $q'_1, \ldots, q'_k, q' \in Q'$ by $\nu'(q') = \sum_{q \in Q} a_{q'}(q) \odot \nu(q)$ and*

$$\mu'_k(\sigma)_{(q'_1,\ldots,q'_k),q'} = \begin{cases} 1 & \text{, if } a_{q'}(q) = \sum_{(q_1,\ldots,q_k) \in Q^k} a_{q'_1}(q_1) \odot \cdots \odot a_{q'_k}(q_k) \odot \mu_k(\sigma)_{(q_1,\ldots,q_k),q} \\ & \quad \text{for every state } q \in Q, \\ 0 & \text{, otherwise.} \end{cases}$$

In order to shorten notation, let $det^p(M) = (Q', \Sigma, \nu', A, \mu')$ for the rest of this section. In the following we show that the extended power set construction is partial. By this we mean that there exists a bu-w-fta such that the extended

power set construction might generate an infinite device (cf. Example 4.6). If the extended power set construction outputs a (finite) bu-w-fta as in Example 4.5, then by Observation 4.2 and Lemma 4.4 the generated automaton is a deterministic device being equivalent to the given automaton.

Observation 4.2. *The tuple $det^p(M)$ is a deterministic bu-w-ta.*

Now we prove that, if $det^p(M)$ is a finite automaton, then M and $det^p(M)$ are equivalent bu-w-fta, which is stated in Lemma 4.4. This proof requires the following preparing lemma.

Lemma 4.3. *Let $k \in \mathbb{N}$ be a non-negative integer, $\sigma \in \Sigma^{(k)}$ a k-ary input symbol, and $q'_1, \ldots, q'_k, q' \in Q'$ states. Then $\mu'_k(\sigma)_{(q'_1,\ldots,q'_k),q'} = \mathbf{1}$, if and only if there exist trees $t_1, \ldots, t_k \in T_\Sigma$ such that $q' = q'_{\sigma(t_1,\ldots,t_k)}$ and $q'_i = q'_{t_i}$ for every index $i \in [k]$.*

Proof. Since $q'_1, \ldots, q'_k, q' \in Q'$ are states of the constructed device, there exist trees $t_1, \ldots, t_k \in T_\Sigma$ such that $q'_i = q'_{t_i}$ for every index $i \in [k]$. Hence

$$
\begin{aligned}
&\mu'_k(\sigma)_{(q'_1,\ldots,q'_k),q'} = \mathbf{1} \\
&\iff (\forall q \in Q) : a_{q'}(q) = \sum_{(q_1,\ldots,q_k)\in Q^k} a_{q'_1}(q_1) \odot \cdots \odot a_{q'_k}(q_k) \odot \mu_k(\sigma)_{(q_1,\ldots,q_k),q} \\
&\iff (\forall q \in Q) : a_{q'}(q) = \sum_{(q_1,\ldots,q_k)\in Q^k} h_\mu(t_1)_{q_1} \odot \cdots \odot h_\mu(t_k)_{q_k} \odot \mu_k(\sigma)_{(q_1,\ldots,q_k),q} \\
&\iff (\forall q \in Q) : a_{q'}(q) = h_\mu(\sigma(t_1,\ldots,t_k))_q \\
&\iff q' = q'_{\sigma(t_1,\ldots,t_k)},
\end{aligned}
$$

which proves the lemma. □

Lemma 4.4. *If the set Q' of states of $det^p(M)$ is finite, then M and $det^p(M)$ are equivalent bu-w-fta.*

Proof. The proof uses the following statement, which we denote by (*): if Q' is a finite set, then for every tree $t \in T_\Sigma$ and state $q' \in Q'$ it holds that $h_{\mu'}(t)_{q'} = \mathbf{1}$, if $q' = q'_t$, and $h_{\mu'}(t)_{q'} = \mathbf{0}$ otherwise. Let us show by induction on the structure of t that (*) holds. Note that the induction base is covered by the induction step. Let $t = \sigma(t_1, \ldots, t_k)$ for some non-negative integer $k \in \mathbb{N}$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, and trees $t_1, \ldots, t_k \in T_\Sigma$. Then

$$
\begin{aligned}
h_{\mu'}(t)_{q'_t} &= \sum_{(q'_1,\ldots,q'_k)\in(Q')^k} h_{\mu'}(t_1)_{q'_1} \odot \cdots \odot h_{\mu'}(t_k)_{q'_k} \odot \mu'_k(\sigma)_{(q'_1,\ldots,q'_k),q'_t} \\
&= h_{\mu'}(t_1)_{q'_{t_1}} \odot \cdots \odot h_{\mu'}(t_k)_{q'_{t_k}} \odot \mu'_k(\sigma)_{(q'_{t_1},\ldots,q'_{t_k}),q'_t} \\
&\qquad \text{(by ind. hyp., } h_{\mu'}(t_i)_{q'_i} = \mathbf{0} \text{ whenever } q'_i \neq q'_{t_i} \text{ for every } i \in [k]\text{)} \\
&= \mathbf{1}. \\
&\qquad \text{(by induction hypothesis and Lemma 4.3)}
\end{aligned}
$$

Hence $q'_t \in \widetilde{\mu'}(t)$. Since $det^p(M)$ is a deterministic bu-w-fta by assumption (finite) and Observation 4.2 (deterministic), we deduce from Observation 3.5(v) that $\widetilde{\mu'}(t) = q'_t$. Thus by Observation 3.5(i) $h_{\mu'}(t)_{q'} = \mathbf{0}$ for every state $q' \in Q \setminus \{q'_t\}$.

Let us now prove the equivalence of the devices M and $det^p(M)$ provided that the set Q' of states of the constructed device is finite. For every tree $t \in T_\Sigma$ the following holds:

$$\begin{aligned}(S_{det^p(M)}, t) &= \sum_{q' \in Q'} h_{\mu'}(t)_{q'} \odot \nu'(q') =_{(\text{by } (*))} h_{\mu'}(t)_{q'_t} \odot \nu'(q'_t) \\ &=_{(\text{by } (*))} \mathbf{1} \odot \sum_{q \in Q} h_\mu(t)_q \odot \nu(q) = (S_M, t).\end{aligned}$$

Thus M and $det^p(M)$ are equivalent bu-w-fta provided the latter device is a finite automaton. □

Let us now present an example of the extended power set construction.

Example 4.5 ($det^p(M)$ **is a finite bu-w-fta).** Consider the bu-w-fta $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, which is given by $Q = \{q_\alpha, q_\perp\}$, $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$, $\nu(q_\alpha) = 1$, $\nu(q_\perp) = 0$, $\mathcal{A} = Bool$, and

$$\mu_0(\alpha) = \begin{array}{c} \begin{array}{cc} q_\alpha & q_\perp \end{array} \\ \begin{pmatrix} 1 & 1 \end{pmatrix} \end{array}, \quad \mu_0(\beta) = \begin{array}{c} \begin{array}{cc} q_\alpha & q_\perp \end{array} \\ \begin{pmatrix} 0 & 1 \end{pmatrix} \end{array}, \quad \mu_2(\sigma) = \begin{array}{c} \begin{array}{cc} q_\alpha & q_\perp \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} (q_\alpha, q_\alpha) \\ (q_\alpha, q_\perp) \\ (q_\perp, q_\alpha) \\ (q_\perp, q_\perp) \end{array}$$

It is easily seen that $h_\mu(t) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and thus $(S_M, t) = 1$, if the input tree t contains a node labeled with α and $h_\mu(t) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and hence $(S_M, t) = 0$, if it does not. It follows that the set of states of $det^p(M)$ is given by $Q' = \{q'_\alpha, q'_\beta\}$. Moreover, $\nu'(q'_\alpha) = 1$, $\nu'(q'_\beta) = 0$, and

$$\mu'_0(\alpha) = \begin{array}{c} \begin{array}{cc} q'_\alpha & q'_\beta \end{array} \\ \begin{pmatrix} 1 & 0 \end{pmatrix} \end{array}, \quad \mu'_0(\beta) = \begin{array}{c} \begin{array}{cc} q'_\alpha & q'_\beta \end{array} \\ \begin{pmatrix} 0 & 1 \end{pmatrix} \end{array}, \quad \mu'_0(\sigma) = \begin{array}{c} \begin{array}{cc} q'_\alpha & q'_\beta \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} (q'_\alpha, q'_\alpha) \\ (q'_\alpha, q'_\beta) \\ (q'_\beta, q'_\alpha) \\ (q'_\beta, q'_\beta) \end{array}$$

Clearly, $det^p(M)$ is a deterministic bu-w-fta. Moreover, a straightforward calculation shows that $h_{\mu'}(t) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and thus $(S_{M'}, t) = 1$, if the input tree t contains a node labeled with α and $h_{\mu'}(t) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and hence $(S_{M'}, t) = 0$, if it does not. Thereby we have shown that the automata M and $det^p(M)$ are equivalent.

Let us now investigate, under which conditions $det^p(M)$ is a finite automaton. We observe that $det^p(M)$ is a (finite) bu-w-fta, if and only if the set $\{h_\mu(t) \mid t \in T_\Sigma\}$ of characteristic vectors is of finite cardinality. In Example 4.5 we have presented a bu-w-fta such that the extended power set construction generates a finite device. Unfortunately there even exists a deterministic bu-w-fta such that the extended power set construction produces an infinite bu-w-ta.

Example 4.6 ($det^p(M)$ **is an infinite bu-w-fta).** Consider the deterministic bu-w-fta $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, which is given by $Q = \{q\}$, $\Sigma = \{\alpha^{(0)}, \beta^{(0)}, \sigma^{(2)}\}$, $\nu(q) = 0$, $\mathcal{A} = Trop$, and $\mu_0(\alpha) = (1)$, $\mu_0(\beta) = (0)$, and $\mu_2(\sigma) = (0)$. A straightforward inductive proof shows that for every input tree t, $h_\mu(t)_q$ equals the number of nodes of t, which are labeled with α. We observe that for every non-negative integer $n \in \mathbb{N}$ the tree t_n, which is inductively defined by $t_0 = \alpha$ and $t_{i+1} = \sigma(\alpha, t_i)$ for every non-negative integer $i \in \mathbb{N}$, contains precisely $n + 1$ nodes labeled with α. Hence, $\{h_\mu(t) \mid t \in T_\Sigma\}$ is an infinite set and thus $det^p(M)$ is an infinite device.

Let us now consider a bu-w-fta M over a finite semiring. Since A^Q is a finite set and $\{h_\mu(t) \mid t \in T_\Sigma\}$ is a subset of A^Q, $\{h_\mu(t) \mid t \in T_\Sigma\}$ is also a finite set. Hence $det^p(M)$ is a (finite) bu-w-fta. In Lemma 4.7 we extend this result by showing that the determinization of a bu-w-fta over a locally finite semiring generates a finite device.

Lemma 4.7. *If $\mathcal{A}$ is a locally finite semiring, then $det^p(M)$ is a (finite) bu-w-fta.*

Proof. Let $A' = \{\mu_k(\sigma)_{(q_1,\ldots,q_k),q} \mid k \in \mathbb{N}, \sigma \in \Sigma^{(k)}, q_1, \ldots, q_k, q \in Q\}$, which is a finite set. Clearly, $h_\mu(t)_q \in \langle A' \rangle_{\{\oplus,\odot\}}$ for every input tree $t \in T_\Sigma$ and state q. Thus, $\{h_\mu(t) \mid t \in T_\Sigma\}$ is a subset of $(\langle A' \rangle_{\{\oplus,\odot\}})^Q$, which by local finiteness is finite. Hence $\{h_\mu(t) \mid t \in T_\Sigma\}$ is a finite set and thus $det^p(M)$ is a (finite) bu-w-fta. □

The following theorem is a consequence of Observation 4.2, Lemma 4.4, and Lemma 4.7. It provides a sufficient condition, under which deterministic and nondeterministic bu-w-fta are equally powerful.

Theorem 4.8 (Sufficient condition for $\mathcal{A}^{d,bu}\langle\langle T_\Sigma \rangle\rangle = \mathcal{A}^{n,bu}\langle\langle T_\Sigma \rangle\rangle$**).** *If $\mathcal{A}$ is a locally finite semiring, then $\mathcal{A}^{d,bu}\langle\langle T_\Sigma \rangle\rangle = \mathcal{A}^{n,bu}\langle\langle T_\Sigma \rangle\rangle$.*

Proof. The claim follows from Lemma 4.7. □

5 Pumping Lemma

In this section we prove a pumping lemma for recognizable tree series. Since there is a one-to-one correspondence between recognizable tree languages and recognizable tree series over the Boolean semiring, we thereby generalize the pumping lemma for classical tree languages, which is (cf. [GS84], Lemma 10.1 of Chapter 2): for every recognizable tree language $L \subseteq T_\Sigma$ there exists an integer $m \in \mathbb{N}$ such that for every tree $t \in L$ with height$(t) \geq m$ there exist contexts $C, C' \in C_\Sigma(X_1)$ and an input tree $s \in T_\Sigma$ such that

(i) $t = C'[C[s]]$,

(ii) $\text{height}(C) > 1$ (i.e., $C \neq x_1$), and

(iii) $C'[C^n[s]] \in L$ for every non-negative integer $n \in \mathbb{N}$.

Similar to classical theories we first show a pumping lemma for tree series, which are accepted by deterministic bu-w-fta. By applying Theorem 4.8 we obtain a result on recognizable tree series. Recall that in classical theories for every device there exists an equivalent deterministic one, while for bu-w-fta a similar result only holds under the assumption that the underlying semiring is locally finite. Hence, the general version of the pumping lemma, which we prove in this section, assumes a locally finite semiring.

Let us start this section by proving a relationship between the characteristic vectors with respect to a deterministic bu-w-fta of the two trees $s, C[s] \in T_\Sigma$, where the latter tree is obtained from the first one by plugging it into a context $C \in C_\Sigma(X_1)$. Since a "run" on the tree $C[s]$ can be decomposed into a "run" on the tree s and a "run" on the context C, one might assume that $h_\mu(C[s])_q$ is the product of the "run" (if it exists) on the input tree s, which has a weight different from $\mathbf{0}$ and ends up in some state p, and the "run" on the context C, which starts at the variable x_1 in state p and which ends in state q. Indeed, such a result was proven in [Bor03], Theorem 1, for commutative semirings. We prove a similar result without the assumption of $\mathcal{A}$ being commutative. For this purpose let us define the two mappings $a_M, b_M : Q \times Q \times C_\Sigma(X_1) \longrightarrow A$, which are inductively defined for every two states $p, q \in Q$ by

$$a_M(p,q,C) = b_M(p,q,C) = \begin{cases} \mathbf{1} & \text{, if } p = q, \\ \mathbf{0} & \text{, otherwise,} \end{cases}$$

provided that C is the trivial context x_1, and

$$a_M(p,q,C) = \begin{cases} h_\mu(t_1)_{\widetilde{\mu}(t_1)} \odot \cdots \odot h_\mu(t_{i-1})_{\widetilde{\mu}(t_{i-1})} \odot a_M(p,r,C') \\ \quad \text{, if } \widetilde{\mu}(t_j) \in Q \text{ for every } j \in [k] \setminus \{i\} \text{ and exists a } r \in Q \text{ such} \\ \quad \text{that } r = \widetilde{\mu}(C'[t]) \text{ for every } t \in T_\Sigma \text{ with } p = \widetilde{\mu}(t), \\ \mathbf{0} \quad \text{, otherwise,} \end{cases}$$

$$b_M(p,q,C) = \begin{cases} b_M(p,r,C') \odot h_\mu(t_{i+1})_{\widetilde{\mu}(t_{i+1})} \odot \cdots \odot h_\mu(t_k)_{\widetilde{\mu}(t_k)} \odot \mu_k(\sigma)_{\vec{q},q} \\ \quad \text{, if } \widetilde{\mu}(t_j) \in Q \text{ for every } j \in [k] \setminus \{i\}, \text{ exists } r \in Q \text{ such} \\ \quad \text{that } r = \widetilde{\mu}(C'[t]) \text{ for every } t \in T_\Sigma \text{ with } p = \widetilde{\mu}(t) \text{ and} \\ \quad \vec{q} = (\widetilde{\mu}(t_1), \ldots, \widetilde{\mu}(t_{i-1}), r, \widetilde{\mu}(t_{i+1}), \ldots, \widetilde{\mu}(t_k)), \\ \mathbf{0} \quad \text{, otherwise,} \end{cases}$$

if $C = \sigma(t_1, \ldots, t_{i-1}, C', t_{i+1}, \ldots, t_k)$ for some positive integers $k \in \mathbb{N}_+$ and $i \in [k]$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, context $C' \in C_\Sigma(X_1)$, and input trees $t_j \in T_\Sigma$ for every index $j \in [k] \setminus \{i\}$.

Lemma 5.1. *Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ be a deterministic bu-w-fta. For every tree $s \in T_\Sigma$ with $\widetilde{\mu}(s) \neq \emptyset$, state $q \in Q$, and context $C \in C_\Sigma(X_1)$ the equation $h_\mu(C[s])_q = a_M(\widetilde{\mu}(s), q, C) \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), q, C)$ holds.*

Proof. The proof is by induction on the length $n \in \mathbb{N}$ of the path from the root of C to the node labeled with the variable x_1 in the context C.

Induction base: Let $n = 0$, i.e., C is the trivial context x_1. Clearly, $h_\mu(C[s])_q = h_\mu(s)_q = a_M(\widetilde{\mu}(s), q, C) \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), q, C)$.

Induction step: Now let $n \geq 1$, i.e., $C = \sigma(t_1, \ldots, t_{i-1}, C', t_{i+1}, \ldots, t_k)$ for some positive integers $k \in \mathbb{N}_+$ and $i \in [k]$, k-ary input symbol $\sigma \in \Sigma^{(k)}$, context $C' \in C_\Sigma(X_1)$, and input trees $t_j \in T_\Sigma$ for every index $j \in [k] \setminus \{i\}$. Let us prove the claimed statement by a case analysis.

Case 1: There exists an index $j \in [k] \setminus \{i\}$ such that $\widetilde{\mu}(t_j) = \emptyset$. Clearly, $\mathbf{0} = a_M(p, q, C) = b_M(p, q, C)$ for every states $p, q \in Q$. By Observation 3.5(iii) also $\widetilde{\mu}(C[s]) = \emptyset$. Moreover, by Observation 3.5(i) $h_\mu(C[s])_q = \mathbf{0} = a_M(\widetilde{\mu}(s), q, C) \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), q, C)$ for every $q \in Q$.

Case 2: We have $\widetilde{\mu}(C'[s]) = \emptyset$. By Observation 3.5(iii) it holds that $\widetilde{\mu}(C[s]) = \emptyset$ and by Observation 3.5(i) we have $h_\mu(C[s])_q = \mathbf{0}$ for every state $q \in Q$. On the other hand, $a_M(\widetilde{\mu}(s), q, C') = \mathbf{0} = b_M(\widetilde{\mu}(s), q, C')$ and thus $h_\mu(C[s])_q = \mathbf{0} = a_M(\widetilde{\mu}(s), q, C) \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), q, C)$ for every state $q \in Q$.

Case 3: The sets $\widetilde{\mu}(C'[s])$ and $\widetilde{\mu}(t_j)$ are singletons for every index $j \in [k] \setminus \{i\}$. In order to shorten notation let $\vec{q} = (\widetilde{\mu}(t_1), \ldots, \widetilde{\mu}(t_{i-1}), \widetilde{\mu}(C'[s]), \widetilde{\mu}(t_{i+1}), \ldots, \widetilde{\mu}(t_k))$. From Observation 3.5(iv) we deduce that $\widetilde{\mu}(C'[s]) = \widetilde{\mu}(C'[t])$ for every tree $t \in T_\Sigma$ with $\widetilde{\mu}(s) = \widetilde{\mu}(t)$, i.e., $a_M(\widetilde{\mu}(s), q, C) = h_\mu(t_1)_{\widetilde{\mu}(t_1)} \odot \cdots \odot h_\mu(t_{i-1})_{\widetilde{\mu}(t_{i-1})} \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[s]), C')$ and $b_M(\widetilde{\mu}(s), q, C) = b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[s]), C') \odot h_\mu(t_{i+1})_{\widetilde{\mu}(t_{i+1})} \odot \cdots \odot h_\mu(t_k)_{\widetilde{\mu}(t_k)} \odot \mu_k(\sigma)_{\vec{q},q}$. Hence, for every state $q \in Q$,

$$
\begin{aligned}
&h_\mu(C[s])_q \\
&= \sum_{(q_1,\ldots,q_k)\in Q^k} h_\mu(t_1)_{q_1} \odot \cdots \odot h_\mu(t_{i-1})_{q_{i-1}} \odot h_\mu(C'[s])_{q_i} \odot h_\mu(t_{i+1})_{q_{i+1}} \odot \cdots \odot \\
&\qquad \odot h_\mu(t_k)_{q_k} \odot \mu_k(\sigma)_{(q_1,\ldots,q_k),q} \\
&= h_\mu(t_1)_{\widetilde{\mu}(t_1)} \odot \cdots \odot h_\mu(t_{i-1})_{\widetilde{\mu}(t_{i-1})} \odot h_\mu(C'[s])_{\widetilde{\mu}(C'[s])} \odot h_\mu(t_{i+1})_{\widetilde{\mu}(t_{i+1})} \odot \cdots \odot \\
&\quad \odot h_\mu(t_k)_{\widetilde{\mu}(t_k)} \odot \mu_k(\sigma)_{\vec{q},q} \\
&\qquad \text{(by Observation 3.5(i) and (v))} \\
&= h_\mu(t_1)_{\widetilde{\mu}(t_1)} \odot \cdots \odot h_\mu(t_{i-1})_{\widetilde{\mu}(t_{i-1})} \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[s]), C') \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot \\
&\quad \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[s]), C') \odot h_\mu(t_{i+1})_{\widetilde{\mu}(t_{i+1})} \odot \cdots \odot h_\mu(t_k)_{\widetilde{\mu}(t_k)} \odot \mu_k(\sigma)_{\vec{q},q} \\
&\qquad \text{(by induction hypothesis)} \\
&= a_M(\widetilde{\mu}(s), q, C) \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), q, C).
\end{aligned}
$$

We note that the above case analysis is complete by Observation 3.5(v). □

Now we show, how pumping a context is reflected in the weights of the runs.

Lemma 5.2. *Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ be a deterministic bu-w-fta, $C \in C_\Sigma(X_1)$ a context, and $s \in T_\Sigma$ a tree such that $\widetilde{\mu}(s) = \widetilde{\mu}(C[s]) \in Q$ is a state. For every non-negative integer $n \in \mathbb{N}$ the equations*

(i) $\widetilde{\mu}(s) = \widetilde{\mu}(C^n[s]) \in Q$ *and*

(ii) $h_\mu(C^n[s])_{\widetilde{\mu}(s)} = a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n$.

hold.

Proof. Claim (i) follows by a repeated application of Corollary 3.6(ii). Let us now prove claim (ii) by induction on $n \in \mathbb{N}$. By Lemma 5.1 the equation $h_\mu(C[t])_q = a_M(\widetilde{\mu}(t), q, C) \odot h_\mu(t)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(t), q, C)$ holds for every state $q \in Q$ and tree $t \in T_\Sigma$ with $\widetilde{\mu}(t) \in Q$.

Induction base: If $n = 0$, then $h_\mu(C^0[s])_{\widetilde{\mu}(s)} = h_\mu(s)_{\widetilde{\mu}(s)} = a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^0 \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^0$.

Induction step: Now assume $n \geq 1$. The following computation proves the claim.

$$
\begin{aligned}
& h_\mu(C^n[s])_{\widetilde{\mu}(s)} \\
&= h_\mu(C[C^{n-1}[s]])_{\widetilde{\mu}(s)} \\
&= a_M(\widetilde{\mu}(C^{n-1}[s]), \widetilde{\mu}(s), C) \odot h_\mu(C^{n-1}[s])_{\widetilde{\mu}(C^{n-1}[s])} \odot b_M(\widetilde{\mu}(C^{n-1}[s]), \widetilde{\mu}(s), C) \\
&\qquad \text{(by Lemma 5.1)} \\
&= a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot h_\mu(C^{n-1}[s])_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \\
&\qquad \text{(by Claim (i))} \\
&= a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot \big(a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^{n-1} \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^{n-1}\big) \odot \\
&\quad \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \\
&\qquad \text{(by induction hypothesis)} \\
&= a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n
\end{aligned}
$$

□

The following lemma assumes a decomposition of a given tree $t \in T_\Sigma$ into contexts $C, C' \in C_\Sigma(X_1)$ and a tree $s \in T_\Sigma$ such that there are "runs" on the trees s and $C[s]$ with a non-zero weight, which end up in the same state. In Lemma 5.4 we prove that such a decomposition exists provided that the input tree satisfies an assumption, which is related the height of the input tree.

Lemma 5.3. *Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ be a deterministic bu-w-fta, $C, C' \in C_\Sigma(X_1)$ contexts, and $s \in T_\Sigma$ an input tree such that $\widetilde{\mu}(s) = \widetilde{\mu}(C[s]) \in Q$ and $\widetilde{\mu}(C'[C[s]]) \in Q$ are states. For every non-negative integer $n \in \mathbb{N}$ it holds that*

$$
\begin{aligned}
h_\mu(C'[C^n[s]])_{\widetilde{\mu}(C'[C^n[s]])} \;=\; & a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot \\
& \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot \\
& \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C').
\end{aligned}
$$

Proof. First we observe that by a repeated application of Corollary 3.6(i) it holds that $\widetilde{\mu}(C^n[s]) = \widetilde{\mu}(C[s]) \in Q$ and thus $\widetilde{\mu}(C'[C^n[s]]) = \widetilde{\mu}(C'[C[s]]) \in Q$. Let us denote this last equation, which shows that the claimed equations are well-defined, by (*). Moreover,

$$
\begin{aligned}
& h_\mu(C'[C^n[s]])_{\widetilde{\mu}(C'[C^n[s]])} \\
&\quad = a_M(\widetilde{\mu}(C^n[s]), \widetilde{\mu}(C'[C^n[s]]), C') \odot h_\mu(C^n[s])_{\widetilde{\mu}(C^n[s])} \odot \\
&\qquad \odot b_M(\widetilde{\mu}(C^n[s]), \widetilde{\mu}(C'[C^n[s]]), C') \\
&\qquad\quad \text{(by Lemma 5.1)} \\
&\quad = a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot h_\mu(C^n[s])_{\widetilde{\mu}(s)} \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \\
&\qquad\quad \text{(by (*))} \\
&\quad = a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot \\
&\qquad \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C'). \\
&\qquad\quad \text{(by Lemma 5.2(ii))}
\end{aligned}
$$

□

Next we show that a decomposition of an input tree $t = C'[C[s]]$ into contexts $C, C' \in C_\Sigma(X_1)$ and a tree $s \in T_\Sigma$ such that $\widetilde{\mu}(s) = \widetilde{\mu}(C[s])$ exists provided there exists a path w in t with more nodes than the given bu-w-fta has states. This can be ensured by requiring $\text{length}(w) \geq \text{card}(Q)$.

Lemma 5.4. *Let M be a deterministic bu-w-fta and $t \in T_\Sigma$ an input tree such that $\widetilde{\mu}(t) \in Q$. For every path $w = w_1 \dots w_{\text{length}(w)} \in \text{paths}(t)$ of t with $w_1, \dots, w_{\text{length}(w)} \in \mathbb{N}$ and $\text{length}(w) \geq \text{card}(Q)$ there exist indices $i, j \in [0, \text{length}(w)]$ such that $i < j$ and $\widetilde{\mu}(t/(w_1 \dots w_i)) = \widetilde{\mu}(t/(w_1 \dots w_j)) \in Q$.*

Proof. Let us start with two observations. For every index $i \in [0, \text{length}(w)]$ there exists a context $C \in C_\Sigma(X_1)$ such that $t = C[t/(w_1 \dots w_i)]$. By Corollary 3.6(ii) and since $\widetilde{\mu}(t) \neq \emptyset$, also $\widetilde{\mu}(t/(w_1 \dots w_i)) \neq \emptyset$. Moreover, by Observation 3.5(v) it holds that $\widetilde{\mu}(t/(w_1 \dots w_i)) \in Q$. The claim now follows from the pigeon hole principle. □

Before we state a smooth version of the pumping lemma for recognizable tree series, let us present a more powerful version of the pumping lemma. A last intermediate result is proven in Lemma 5.5.

Lemma 5.5. *Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ be a deterministic bu-w-fta. For every input tree $t \in T_\Sigma$ such that $\widetilde{\mu}(t) \in Q$ is a state and for every path $w = w_1 \dots w_l \in \text{paths}(t)$ of t with $w_1, \dots, w_{\text{length}(w)} \in \mathbb{N}$ and $\text{length}(w) = l \geq \text{card}(Q)$ there exist contexts $C, C' \in C_\Sigma(X_1)$ and an input tree $s \in T_\Sigma$ such that*

(i) $t = C'[C[s]]$,

(ii) there exist indices $i, j \in [0, l]$ with $i < j$ and $l - i \leq \text{card}(Q)$ such that $s = t/(w_1 \dots w_j)$ and $C[s] = t/w_1 \dots w_i$,

(iii) $\widetilde{\mu}(C^n[s]) = \widetilde{\mu}(s) \in Q$ *and* $\widetilde{\mu}(C'[C^n[s]]) = \widetilde{\mu}(t) \in Q$ *for every non-negative integer* $n \in \mathbb{N}$, *and*

(iv) for every non-negative integer $n \in \mathbb{N}$ *it holds that*

$$\begin{aligned} &h_\mu(C'[C^n[s]])_{\widetilde{\mu}(C'[C^n[s]])} \\ &= a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot \\ &\quad \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C'). \end{aligned}$$

Proof. Clearly, $w_{l-\mathrm{card}(Q)+1} \dots w_l$ is a path of length $\mathrm{card}(Q)$ of the tree $t/(w_1 \dots w_{l-\mathrm{card}(Q)})$. Since $\widetilde{\mu}(t) \in Q$, we deduce from Corollary 3.6(i) that $\widetilde{\mu}(t/w_1 \dots w_{l-\mathrm{card}(Q)}) \neq \emptyset$ and hence by Lemma 5.4 there exist indices $i', j' \in [0, \mathrm{card}(Q)]$ with $i' < j'$ such that

$$\begin{aligned} &\widetilde{\mu}(t/(w_1 \dots w_{l-\mathrm{card}(Q)+i'})) \\ &\qquad = \widetilde{\mu}((t/(w_1 \dots w_{l-\mathrm{card}(Q)}))/(w_{l-\mathrm{card}(Q)+1} \dots w_{l-\mathrm{card}(Q)+i'})) \\ &\qquad = \widetilde{\mu}((t/(w_1 \dots w_{l-\mathrm{card}(Q)}))/(w_{l-\mathrm{card}(Q)+1} \dots w_{l-\mathrm{card}(Q)+j'})) \\ &\qquad = \widetilde{\mu}(t/(w_1 \dots w_{l-\mathrm{card}(Q)+j'})). \end{aligned}$$

Let us denote this statement by (*). We set $i = l - \mathrm{card}(Q) + i'$, $j = l - \mathrm{card}(Q) + j'$, and $s = t/(w_1 \dots w_j)$. Moreover, we choose the contexts $C, C' \in C_\Sigma(X_1)$ such that $C[s] = t/w_1 \dots w_i$ and $C'[C[s]] = t$. Then (i) is satisfied. Furthermore, $i, j \in [0, l]$ and $i < j$ by the choice of the indices i' and j' and since $l \geq \mathrm{card}(Q)$. Hence (ii) holds. From Corollary 3.6(ii), the assumption $\widetilde{\mu}(t) \in Q$, and Condition (i) we deduce that $\widetilde{\mu}(s) \in Q$. Thus, also applying (*),

$$\widetilde{\mu}(s) = \widetilde{\mu}(t/(w_1 \dots w_j)) = \widetilde{\mu}(t/(w_1 \dots w_i)) = \widetilde{\mu}(C[s]) \in Q$$

by (*). Hence, by Lemma 5.2(ii) and Corollary 3.6(ii) it holds that $\widetilde{\mu}(C^n[s]) = \widetilde{\mu}(s) \in Q$ and $\widetilde{\mu}(C'[C^n[s]]) = \widetilde{\mu}(t) \in Q$ for every non-negative integer $n \in \mathbb{N}$, which shows Property (iii). Statement (iv) follows from Property (iii) and Lemma 5.3. □

Theorem 5.6 (Pumping Lemma (i)). *Let (a)* $\mathcal{A}$ *be a semiring and* $S \in \mathcal{A}^{d,bu}\langle\langle T_\Sigma \rangle\rangle$ *or (b)* $\mathcal{A}$ *be a locally finite semiring and* $S \in \mathcal{A}^{n,bu}\langle\langle T_\Sigma \rangle\rangle$. *There exists a non-negative integer* $m \in \mathbb{N}$ *such that for every tree* $t \in \mathrm{supp}(S)$ *and path* $w = w_1 \dots w_l \in path(t)$ *of* t *with* $w_1, \dots, w_{\mathrm{length}(w)} \in \mathbb{N}$ *and* $\mathrm{length}(w) = l \geq m$, *there exist contexts* $C, C' \in C_\Sigma(X_1)$, *a tree* $s \in T_\Sigma$, *and semiring elements* $a, a', b, b', c \in A$ *such that*

(i) $t = C'[C[s]]$,

(ii) there exist indices $i, j \in [0, l]$ *with* $i < j$ *and* $l - i \leq m$ *such that* $s = t/w_1 \dots w_j$ *and* $C[s] = t/w_1 \dots w_i$, *and*

(iii) $(S, C'[C^n[s]]) = a' \odot a^n \odot c \odot b^n \odot b'$ *for every non-negative integer* $n \in \mathbb{N}$.

Moreover, if $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ *is a deterministic bu-w-fta, which accepts* S, *then*

(iv) m *can be chosen to be the number of states of* M,

(v) $\widetilde{\mu}(C^n[s]) = \widetilde{\mu}(s) \in Q$ *and* $\widetilde{\mu}(C'[C^n[s]]) = \widetilde{\mu}(t) \in Q$ *for every non-negative integer* $n \in \mathbb{N}$, *and*

(vi) the semiring elements $a, a', b, b', c \in A$ *can be set to*

$$\begin{array}{lcllcl} a & = & a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C), & a' & = & a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C'), \\ b & = & b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C), & b' & = & b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot \nu(C'[C[s]]), \\ c & = & h_\mu(s)_{\widetilde{\mu}(s)}. & & & \end{array}$$

Proof. By Theorem 4.8 we have $\mathcal{A}^{n,bu}\langle\langle T_\Sigma \rangle\rangle = \mathcal{A}^{d,bu}\langle\langle T_\Sigma \rangle\rangle$, if the underlying semiring is locally finite. Hence it suffices to consider a tree series $S \in \mathcal{A}^{d,bu}\langle\langle T_\Sigma \rangle\rangle$, which is accepted by a deterministic device. Let $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ such a bu-w-fta. We set $m = \mathrm{card}(Q)$. Furthermore, let $t \in \mathrm{supp}(S)$ be an input tree and $w = w_1 \dots w_l \in \mathrm{paths}(t)$ a path of t of length $l \geq m$. In particular, $\widetilde{\mu}(t) \in Q$ by Observation 3.5(i), (v), and (vi). By Lemma 5.5 there exist contexts $C, C' \in C_\Sigma(X_1)$ and a tree $s \in T_\Sigma$ such that Properties (i), (ii), (iv), and (v) are satisfied. Moreover, by Observation 3.5(vi) it holds for every non-negative integer $n \in \mathbb{N}$ that

$$\begin{aligned} (S, C'[C^n[s]]) &= h_\mu(C'[C^n[s]])_{\widetilde{\mu}(C'[C^n[s]])} \odot \nu(\widetilde{\mu}(C'[C^n[s]])) \\ &= h_\mu(C'[C^n[s]])_{\widetilde{\mu}(C'[C^n[s]])} \odot \nu(\widetilde{\mu}(C'[C[s]])) \\ &= a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot \\ &\quad \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot \\ &\quad \odot \nu(\widetilde{\mu}(C'[C[s]])), \end{aligned}$$

which proves the remaining Statements (iii) and (vi). □

Let us now present a smoother, but less powerful version of the pumping lemma:

Lemma 5.7. *Let* $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ *be a deterministic bu-w-fta. For every input tree* $t \in T_\Sigma$ *with* $\mathrm{height}(t) \geq \mathrm{card}(Q) + 1$ *and* $\widetilde{\mu}(t) \in Q$ *there exist contexts* $C, C' \in C_\Sigma(X_1)$, *a tree* $s \in T_\Sigma$, *and semiring elements* $a, a', b, b' \in A$ *such that*

(i) $t = C'[C[s]]$,

(ii) $\mathrm{height}(C[s]) \leq \mathrm{card}(Q) + 1$ *and* $C \neq x_1$, *and*

(iii) $\widetilde{\mu}(C^n[s]) = \widetilde{\mu}(s) \in Q$ *and* $\widetilde{\mu}(C'[C^n[s]]) = \widetilde{\mu}(t) \in Q$ *for every non-negative integer* $n \in \mathbb{N}$,

(iv) for every non-negative integer $n \in \mathbb{N}$ *it holds that*

$$\begin{aligned} &h_\mu(C'[C^n[s]])_{\widetilde{\mu}(C'[C^n[s]])} \\ &= a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot \\ &\quad \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)^n \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C'). \end{aligned}$$

Proof. Let $t \in T_\Sigma$ be an input tree of height $\geq \mathrm{card}(Q) + 1$ such that $\widetilde{\mu}(t) \in Q$ and $w = w_1 \dots w_l \in \mathrm{paths}(t)$ be a longest path of t, where $w_1, \dots, w_l \in \mathbb{N}$. By Observation 2.2 it holds that $l \geq \mathrm{card}(Q)$. Hence by Lemma 5.5 there exist contexts $C, C' \in C_\Sigma(X_1)$, a tree $s \in T_\Sigma$ and semiring elements $a, a', b, b', c \in A$ such that (i), (iii), and (iv) are satisfied. Moreover, there exist indices $i, j \in [0, l]$ with $i < j$ and $l - i \leq \mathrm{card}(Q)$ such that $s = t/(w_1 \dots w_j)$ and $C[s] = t/(w_1 \dots w_i)$. We observe that $w_{i+1} \dots w_l$ is a longest path of the tree $C[s]$ of length $l - i$ and thus $\mathrm{height}(C[s]) = l - i + 1 \leq \mathrm{card}(Q) + 1$ by Observation 2.2. Furthermore, C is not the trivial context x_1 by $i < j$, $s = t/(w_1 \dots w_j)$, and $C[s] = t/(w_1 \dots w_i)$. Thus also claim (ii) holds. □

Corollary 5.8 (Pumping Lemma (ii)). *Let (a) $\mathcal{A}$ be a semiring and $S \in \mathcal{A}^{d,bu}\langle\langle T_\Sigma \rangle\rangle$ or (b) $\mathcal{A}$ be a locally finite semiring and $S \in \mathcal{A}^{n,bu}\langle\langle T_\Sigma \rangle\rangle$. There exists a non-negative integer $m \in \mathbb{N}$ such that for every tree $t \in \mathrm{supp}(S)$ with $\mathrm{height}(t) \geq m + 1$ there exist contexts $C, C' \in C_\Sigma(X_1)$, a tree $s \in T_\Sigma$, and semiring elements $a, a', b, b', c \in A$ such that*

(i) $t = C'[C[s]]$,

(ii) $\mathrm{height}(C[s]) \leq m + 1$ *and* $C \neq x_1$, *and*

(iii) $(S, C'[C^n[s]]) = a' \odot a^n \odot c \odot b^n \odot b'$ *for every non-negative integer* $n \in \mathbb{N}$.

Furthermore, if $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$ is a deterministic bu-w-fta, which accepts S, then

(iv) m *can be chosen to be the number of states of* M.

(v) $\widetilde{\mu}(C^n[s]) = \widetilde{\mu}(s) \in Q$ *and* $\widetilde{\mu}(C'[C^n[s]]) = \widetilde{\mu}(t) \in Q$ *for every non-negative integer* $n \in \mathbb{N}$,

(vi) the semiring elements $a, a', b, b', c \in A$ *can be set to*

$$\begin{array}{lcllcl} a &=& a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C), & a' &=& a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C'), \\ b &=& b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C), & b' &=& b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[C[s]]), C') \odot \nu(C'[C[s]]) \\ c &=& h_\mu(s)_{\widetilde{\mu}(s)}. & & & \end{array}$$

Proof. The proof is very similar to the proof of Theorem 5.6. We note that it uses Lemma 5.7 rather than Lemma 5.5 and leave it to the reader. □

We note that the two versions of the pumping lemma do not say anything about the support of the considdered tree series unless the underlying semiring is zero-divisor free.

Similar to classical theories the pumping lemma can be applied for proving, that a tree series is not recognizable. Let us give an example for this application. We prove that the tree series, which maps every input tree to its height, is not recognized by a deterministic device over the Arctic semiring. This result is surprising by two reasons: first, the height of a tree is defined in terms of max and +,

which are the operations of the Arctic semiring. Second, the support of this tree series is T_Σ, which is recognizable by a classical one-state (and hence deterministic) bottom-up finite tree automata. Thus, associating weights to a recognizable tree language, might destroy recognizability.

Example 5.9 ($\sum_{t \in T_\Sigma}$ height(t) t not acceptable by deterministic bu-w-fta). Let $(S, t) = \text{height}(t)$ for every input tree $t \in T_\Sigma$, where the underlying ranked alphabet is given by $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$ and the underlying algebraic structure is the Arctic semiring. We show by contradiction that S is not recognizable by a deterministic bu-w-fta. Assume the converse. By Theorem 5.6 there exists a non-negative integer $m \in \mathbb{N}$ such that for every tree $t \in \text{supp}(S)$ and path $w = w_1 \dots w_l \in \text{paths}(t)$ of t with $w_1, \dots, w_l \in \mathbb{N}$ and length$(w) = l \geq m$ there exist contexts $C, C' \in C_\Sigma(X_1)$, a tree $s \in T_\Sigma$, and semiring elements $a, a', b, b', c \in A$ such that (i) $t = C'[C[s]]$, (ii) there exist indices $i, j \in [0, l]$ with $i < j$ and $l - i \leq m$ such that $s = t/w_1 \dots w_j$ and $C[s] = t/w_1 \dots w_i$, and (iii) height$(C'[C^n[s]]) = a' + a^n + c + b^n + b'$ for every non-negative integer $n \in \mathbb{N}$. Let

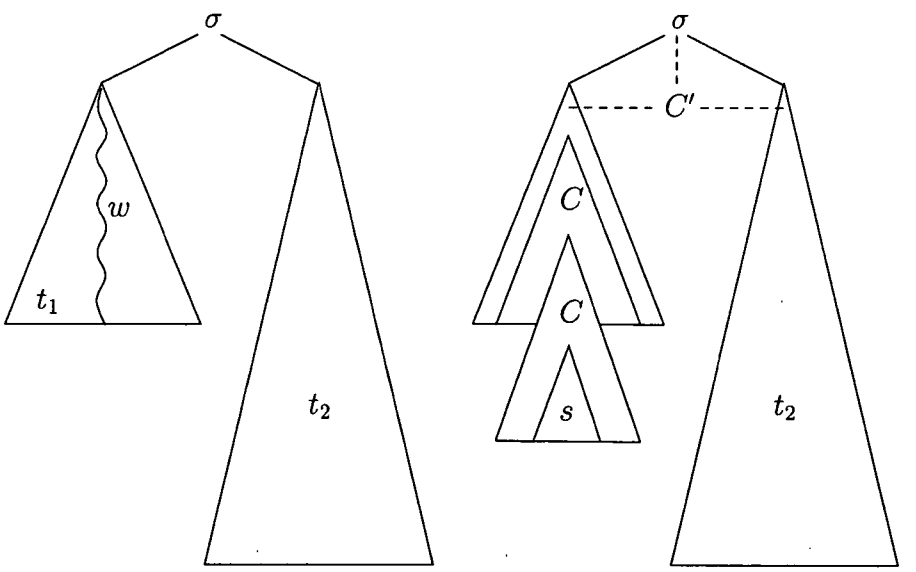


Figure 1: The trees $t = C'[C[s]]$ and $C'[C^2[s]]$.

us consider the particular tree $t = \sigma(t_1, t_2)$, where $t_1, t_2 \in T_\Sigma$ are trees of height $m + 1$ and $2m + 1$, respectively. Hence, $(S, t) = \text{height}(t) = 2m + 2$. Moreover, there exists a path $w = w_1 w_2 \dots w_{m+1}$ with $w_1 = 1$ of the input tree t of length $l = m + 1$ such that $w_2 \dots w_{m+1}$ is a (longest) path of the subtree t_1. Hence, there exists a decomposition of t along the path w satisfying (i), (ii), and (iii). Since w is a path of length $l = m + 1$, we deduce from (ii) that $i \geq 1$. Hence $w_1 \dots w_i \neq \varepsilon$ and $C[s]$ is a subtree of t_1 (cf. Figure 1). We now show that pumping the context C once, does not increase the height of the tree: to do so let us consider the paths of

the pumped tree $C'[C^2[s]]$. Every such path $w \in \text{paths}(C'[C^2[s]])$ is either a path of the given tree $t = C'[C[s]]$ (i.e., its length is $\leq 2m+1$) or $w = 1.w_2 \dots w_i \dots w_j \overline{w}$ for some path $\overline{w}$ of the tree $C[s]$. Since $C[s]$ is a subtree of t_1, we deduce from Observation 2.2 that $\text{length}(\overline{w}) \leq \text{height}(t_1) - 1 = m$. Moreover, $j \leq m+1$ by (ii), thus $\text{length}(w) \leq j + \text{length}(\overline{w}) \leq 2m+1$. Hence every path of the pumped tree $C'[C^2[s]]$ is of length $\leq 2m+1$. Thus $\text{height}(C'[C[s]]) = \text{height}(C'[C^2[s]]) = 2m+2$. From (iii) we now deduce that $a' + a + c + b + b' = 2m+2 = a' + 2a + c + 2b + b'$, i.e., $a = b = 0$. But then $\text{height}(C'[C^n[s]]) = a' + n \cdot a + c + n \cdot b + b' = 2m+2$ for every non-negative integer $n \in \mathbb{N}$, which means that pumping the context C arbitrary often does not increase the height of the pumped tree. Hence C must be the trivial context x_1. But this contradicts to (ii), since $t/w_1 \dots w_j = s = C[s] = t/w_1 \dots w_i$ and thus $i = j$. Hence S is not recognizable by a deterministic bu-w-fta over the Arctic semiring.

6 Decidability

In this section we investigate the following decidability problems for a given recognizable tree series $S \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$:

- **Constant-on-its-support problem:** Is the tree series S constant on its support?
- **Constant problem:** Is S a constant tree series?
- **Boolean problem:** Is S a boolean tree series?
- **Emptiness problem:** Is the support of S the empty set?
- **Finiteness problem:** Is the support of S finite?

Note that throughout this section decidable stands for effectively decidable and computable stands for effectively computable (in the sense of Church's hypothesis, cf. [HU79]). It turns out that the aforementioned five problems are decidable, provided that the given tree series S is recognized by a deterministic bu-w-fta over a commutative semiring. The decidability of the Finiteness problem is proven for zero-divisor free and commutative semirings. When talking about decidability problems, it is a common assumption to consider finitely represented input. So we do in the present paper. A recognizable tree series can be finitely represented by a bu-w-fta, which accepts this tree series. Hence we assume for the rest of this section, that besides the tree series S also a deterministic bu-w-fta $M = (Q, \Sigma, \nu, \mathcal{A}, \mu)$, which accepts S, is given. Note that by Lemma 4.7 all the results of this section also hold, if M is a non-deterministic bu-w-fta over a locally finite and commutative semiring $\mathcal{A}$. We also assume throughout this section that $\mathcal{A} = (A, \oplus, \odot, \mathbf{0}, \mathbf{1})$ is a computable semiring.

First we prove the decidability of both constant problems by applying the pumping lemma. The decidability of the Emptiness and Boolean problems are straightforward consequences of the decidability of the Constant-on-its-support problem.

Finally we show that the Finiteness problem is decidable by reducing it to the Emptiness problem.

In order to shorten notation let us define the set P, in which we collect all those states, which are reachable by "small" trees s and there exists a context C, the height of which is bounded by $2 \cdot \mathrm{card}(Q) - 1$ such that $C[s] \in \mathrm{supp}(S)$:

$$P = \{\widetilde{\mu}(s) \in Q \mid (\exists s \in T_\Sigma), (\exists C \in C_\Sigma(X_1)) : \mathrm{height}(s) \leq \mathrm{card}(Q), \mathrm{height}(C) \leq 2 \cdot \mathrm{card}(Q) - 1, C[s] \in \mathrm{supp}(S)\}.$$

Observation 6.1. *The set P is computable.*

In the following lemma we show that dropping the requirement on the height of the context C in the definition of P does not effect the set P.

Lemma 6.2. *If $\mathcal{A}$ is a commutative semiring, then it holds that* $P = \{\widetilde{\mu}(s) \in Q \mid (\exists s \in T_\Sigma), (\exists C \in C_\Sigma(X_1)) : \mathrm{height}(s) \leq \mathrm{card}(Q), C[s] \in \mathrm{supp}(S)\}$.

Proof. Let $P' = \{\widetilde{\mu}(s) \in Q \mid (\exists s \in T_\Sigma), (\exists C \in C_\Sigma(X_1)) : \mathrm{height}(s) \leq \mathrm{card}(Q), C[s] \in \mathrm{supp}(S)\}$. Clearly, $P \subseteq P'$. It remains to show $P' \subseteq P$. Let $p \in P'$, i.e., there exist a tree $s \in T_\Sigma$ of height $\leq \mathrm{card}(Q)$ and a context $C \in C_\Sigma(X_1)$ such that $p = \widetilde{\mu}(s)$ and $C[s] \in \mathrm{supp}(S)$. We show by contradiction that then also $p \in P$. Assume that $p \notin P$. Let s be fixed and assume without loss of generality that C is chosen such that $C[s]$ is of minimal size satisfying $C[s] \in \mathrm{supp}(S)$. We set $t = C[s]$. Clearly, $\mathrm{height}(C) \geq 2 \cdot \mathrm{card}(Q)$. In order to apply the pumping lemma (Theorem 5.6) we specify a path $w = w_1 \dots w_l \in \mathrm{paths}(C)$, where $w_1, \dots, w_l \in \mathbb{N}$, as follows (cf. Figure 2). Let $w' \in \mathrm{paths}(C)$ denote the path such that the node of C at w' is labeled with x_1.

(α) If $\mathrm{length}(w') \geq \mathrm{card}(Q) + 1$, then set $w = w'$.

(β) If $\mathrm{length}(w') \leq \mathrm{card}(Q)$, then let w be an arbitrary path of length $2 \cdot \mathrm{card}(Q) + 1$, which exists by $\mathrm{height}(C) \geq 2 \cdot \mathrm{card}(Q)$ and Observation 2.2.

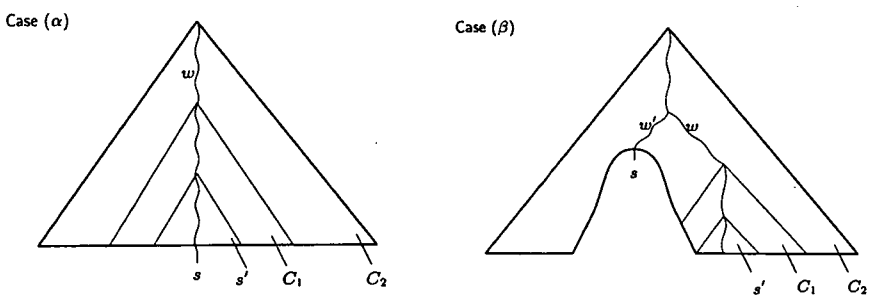


Figure 2: Decomposition of $C[s]$ in Cases (α) and (β).

By the pumping lemma (Theorem 5.6) there exists a decomposition of $t = C_2[C_1[s']]$ along the path w into contexts $C_1, C_2 \in C_\Sigma(X_1)$ and a tree $s' \in T_\Sigma$ such that

(i) there exist indices $i, j \in [0, l]$ with $i < j$ and $l - i \leq \mathrm{card}(Q)$ such that $s' = t/w_1 \dots w_j$ and $C_1[s'] = t/w_1 \dots w_i$, and

(ii) there exist semiring elements $a, a', b, b', c \in A$ such that $(S, C_2[C_1^n[s]]) = a' \odot a^n \odot c \odot b^n \odot b'$ for every non-negative integer $n \in \mathbb{N}$.

From the choice of w it follows that s is a subtree of $C_2[s']$, which is formally proven as follows:

(α) By (i) s is a subtree of s' and thus s is a subtree of $C_2[s']$.

(β) By (i) it holds that $i \geq \mathrm{card}(Q) + 1$ and thus $w_1 \dots w_i$ is not a prefix of w'. Clearly, $\mathrm{paths}(C_2) = \{w \in \mathrm{paths}(C) \mid w_1 \dots w_i \text{ is not a prefix of } w\}$, from which we deduce that $w' \in \mathrm{paths}(C_2)$. Apparently s is a subtree of $C_2[s']$.

Thus there exists a context $C_3 \in C_\Sigma(X_1)$ such that $C_3[s] = C_2[s']$. Moreover, by (i) $C_1 \neq x_1$ and thus $\mathrm{size}(C_3[s]) = \mathrm{size}(C_2[s']) < \mathrm{size}(C_2[C_1[s']]) = \mathrm{size}(t)$. Let us denote this last statement by (‡). From (ii) we deduce that $(S, t) = a \odot b \odot (S, C_2[s']) = a \odot b \odot (S, C_3[s])$. We assumed that $t \in \mathrm{supp}(S)$, thus $(S, t) \neq \mathbf{0}$ and by the aforementioned equation $(S, C_3[s]) \neq \mathbf{0}$, i.e., $C_3[s] \in \mathrm{supp}(S)$. Since the context C was chosen such that $t = C[s]$ is of minimal size violating the statement of the lemma and by (‡) $\mathrm{size}(C_3[s]) < \mathrm{size}(t)$ it holds that $\widetilde{\mu}(s) \in P$. This contradicts to the assumption $\widetilde{\mu}(s) \notin P$. Hence $P = P'$. □

We note that by a very similar proof also the assumption in P on the height of s can be dropped. Since this statement is not needed in the present paper, we leave the proof of this claim to the reader.

The decision procedures will be in terms of two sets B_1 and B_2. Roughly speaking, in B_1 we collect the costs of "small" contexts, which can be pumped, while B_2 is the set of all "small" trees. Formally,

$$B_1 = \{a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \in A \mid (\exists s \in T_\Sigma), (\exists C \in C_\Sigma(X_1)) : \mathrm{height}(C[s]) \leq \mathrm{card}(Q) + 1, \widetilde{\mu}(s) = \widetilde{\mu}(C[s]) \in P\},$$

$$B_2 = \{s \in T_\Sigma \mid \mathrm{height}(s) \leq \mathrm{card}(Q)\}.$$

Lemma 6.3. *Let $\mathcal{A}$ be a commutative semiring and $d \in A$ a semiring element. For every input tree $t \in \mathrm{supp}(S)$ it holds that $(S, t) = d$, if and only if*

(CPS1) for every semiring element $b \in B_1$ it holds that $b \odot d \in \{\mathbf{0}, d\}$ and

(CPS2) for every tree $s \in B_2$ it holds that $(S, s) \in \{\mathbf{0}, d\}$.

Proof. First assume that $(S, t) = d$ for every input tree $t \in \mathrm{supp}(S)$. In particular, $d \neq 0$. We show that (CPS1) and (CPS2) are satisfied. Clearly, (CPS2) holds. Let us now prove that also (CPS1) is fulfilled. Let $b \in B_1$. By the definition of B_1 there exist a tree $s \in T_\Sigma$ and a context $C \in C_\Sigma(X_1)$ with $\mathrm{height}(C[s]) \leq \mathrm{card}(Q) + 1$ such that $b = a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)$ and $\widetilde{\mu}(s) = \widetilde{\mu}(C[s]) \in P$.

(α) If $C = x_1$ is the trivial context, then $a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) = b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) = \mathbf{1}$, thus $b = \mathbf{1}$ and $d \odot b = d \in \{\mathbf{0}, d\}$.

(β) If $C \neq x_1$, then $\mathrm{height}(C[s]) \leq \mathrm{card}(Q) + 1$ implies $\mathrm{height}(s) \leq \mathrm{card}(Q)$. Hence by Lemma 6.2 there exists a context $C' \in C_\Sigma(X_1)$ such that $C'[s] \in \mathrm{supp}(S)$ and thus $(S, C'[s]) = d$. Thus

$$
\begin{aligned}
& b \odot d \\
&= a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot (S, C'[s]) \\
&= a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot h_\mu(C'[s])_{\widetilde{\mu}(C'[s])} \odot \nu(\widetilde{\mu}(C'[s])) \\
&\qquad \text{(by Observation 3.5(vi))} \\
&= a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot a_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[s]), C') \odot \\
&\quad \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C'[s]), C') \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot \nu(\widetilde{\mu}(C'[s])) \\
&\qquad \text{(by Lemma 5.1)} \\
&= a_M(\widetilde{\mu}(s), \widetilde{\mu}(C[s]), C) \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(C[s]), C) \odot \\
&\quad \odot a_M(\widetilde{\mu}(C[s]), \widetilde{\mu}(C'[C[s]]), C') \odot b_M(\widetilde{\mu}(C[s]), \widetilde{\mu}(C'[C[s]]), C') \odot \\
&\quad \odot h_\mu(s)_{\widetilde{\mu}(s)} \odot \nu(\widetilde{\mu}(C'[C[s]])) \\
&\qquad \text{(by } \widetilde{\mu}(s) = \widetilde{\mu}(C[s]) \text{ and Corollary 3.6)} \\
&= h_\mu(C'[C[s]])_{\widetilde{\mu}(C'[C[s]])} \odot \nu(\widetilde{\mu}(C'[C[s]])) \\
&\qquad \text{(by Lemma 5.1)} \\
&= (S, C'[C[s]]) \\
&\qquad \text{(by Observation 3.5(vi))} \\
&\in \{\mathbf{0}, d\}.
\end{aligned}
$$

Hence also (CPS1) is satisfied.

Conversely assume that (CPS1) and (CPS2) hold. We show that $(S, t) = d$ for every input tree $t \in \mathrm{supp}(S)$. Without loss of generality we assume that $d \neq \mathbf{0}$. In fact, if $d = \mathbf{0}$, then it holds that $S = \widetilde{\mathbf{0}}$, i.e., $\mathrm{supp}(S) = \emptyset$ and thus $(S, t) = d$ for every input tree $t \in \mathrm{supp}(S)$ and semiring element $d \in A$. The prove of the claimed implication is by contradiction. In assuming the converse, let $t \in \mathrm{supp}(S)$ be a tree with $(S, t) \neq d$. Without loss of generality let $t \in \mathrm{supp}(S)$ be of minimal size violating $(S, t) = d$. Since by (CPS2) $(S, s) \in \{\mathbf{0}, d\}$ for every tree $s \in T_\Sigma$ of height $\leq \mathrm{card}(Q)$, we deduce $\mathrm{height}(t) \geq \mathrm{card}(Q) + 1$. By the pumping lemma (Corollary 5.8) there exists a decomposition $t = C'[C[s]]$ for some tree $s \in T_\Sigma$ and contexts $C, C' \in C_\Sigma(X_1)$ with $C \neq x_1$, $\mathrm{height}(C[s]) \leq \mathrm{card}(Q) + 1$, and $\widetilde{\mu}(s) = \widetilde{\mu}(C[s])$. Moreover, there exist semiring elements $a, a', b, b', c \in A$ such that $(S, t) = a' \odot a \odot c \odot b \odot b'$ and $(S, C'[s]) = a' \odot c \odot b'$. Consequently, $(S, t) = a \odot b \odot (S, C'[s])$, which we denote by (†). Also by the pumping lemma (Corollary 5.8) a and b can be set to

$$a = a_M(\widetilde{\mu}(s), \widetilde{\mu}(C[s]), C) \quad \text{and} \quad b = b_M(\widetilde{\mu}(s), \widetilde{\mu}(C[s]), C).$$

Since $t = C'[C[s]] \in \mathrm{supp}(S)$ and the fact that $\mathrm{height}(s) < \mathrm{height}(C[s]) \leq \mathrm{card}(Q) + 1$, we deduce from Lemma 6.2 that $\widetilde{\mu}(s) \in P$ and thus $a \odot b \in B_1$. Let us denote this last statement by (‡). Furthermore, $\mathrm{size}(C'[s]) < \mathrm{size}(t)$ by C not being the trivial context x_1. From this and the assumption that $t \in \mathrm{supp}(S)$ is of minimal size violating $(S,t) = d$ we deduce that $(S, C'[s]) \in \{\mathbf{0}, d\}$. Hence by (†), (‡), and Condition (CPS1) it follows that $(S,t) = a \odot b \odot (S, C'[s]) \in \{\mathbf{0}, d\}$, which contradicts to the assumptions $t \in \mathrm{supp}(S)$ and $(S,t) \neq d$. Hence $(S,t) = d$ for every tree $t \in \mathrm{supp}(S)$. □

In order to state the decidability of the Constant-on-its-support problem it remains to show that B_1 and B_2 are computable.

Lemma 6.4. *The sets B_1 and B_2 are computable.*

Proof. The claim is trivial for the set B_2. Let us show that also B_1 is computable. Since P is a computable set by Observation 6.1, also $B = \{(s, C) \in T_\Sigma \times C_\Sigma(X_1) \mid \mathrm{height}(C[s]) \leq \mathrm{card}(Q)+1, \widetilde{\mu}(s) = \widetilde{\mu}(C[s]) \in P\}$ is computable. It remains to show that $a_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C) \odot b_M(\widetilde{\mu}(s), \widetilde{\mu}(s), C)$ is computable for every pair $(s, C) \in B$. Therefore it suffices to show that $a_M(p, q, C)$ and $b_M(p, q, C)$ are computable for every two states $p, q \in Q$ and context $C \in C_\Sigma(X_1)$ of height $\leq \mathrm{card}(Q)+1$. We show the claim for $a_M(p, q, C)$ and note that the proof of $b_M(p, q, C)$ being computable is very similar. The prove of the computability of $a_M(p, q, C)$ is by induction on the structure of C:

Induction base: If C is the trivial context x_1, the claim is trivial.

Induction step: Now let $C = \sigma(t_1, \ldots, t_{i-1}, C', t_{i+1}, \ldots, t_k)$ for some positive integers $k \in \mathbb{N}_+$ and $i \in [k]$, k-ary input symbol $\sigma \in \Sigma^{(k)}$ and trees $t_j \in T_\Sigma$ for every index $j \in [k] \setminus \{i\}$. From $\mathrm{height}(C) \leq \mathrm{card}(Q) + 1$ we deduce that, for every index $j \in [k] \setminus \{i\}$ it holds that $\mathrm{height}(t_j) \leq \mathrm{card}(Q)$. Hence the set $\widetilde{\mu}(t_j)$ is computable and thus it is decidable whether $\widetilde{\mu}(t_j) \in Q$ for every index $j \in [k] \setminus \{i\}$. Next we show that it is decidable whether there exists a state $r \in Q$ such that $r = \widetilde{\mu}(C'[t])$ for every tree $t \in T_\Sigma$ with $p = \widetilde{\mu}(t)$. We also show that this state r is computable, if it exists. It holds for every state $r \in Q$ that:

	For every tree $t \in T_\Sigma$ with $p = \widetilde{\mu}(t)$ it holds that $r = \widetilde{\mu}(C'[t])$.
$\Longleftrightarrow^{(*)}$	There exists a tree $t \in T_\Sigma$ with $p = \widetilde{\mu}(t)$ and $r = \widetilde{\mu}(C'[t])$.
$\Longleftrightarrow^{(**)}$	There exists a tree $t \in T_\Sigma$ of height $\leq \mathrm{card}(Q)$ with $p = \widetilde{\mu}(t)$ and $r = \widetilde{\mu}(C'[t])$.

Note that $\Rightarrow^{(*)}$ follows from the definition of P, $\Leftarrow^{(*)}$ is a consequence of Corollary 3.6(ii), $\Leftarrow^{(**)}$ follows by a repeated application of Corollary 5.8(v) with $n = 0$, and $\Leftarrow^{(**)}$ trivially holds. Clearly, the latter statement is decidable. From the above equivalences we deduce that, if r has the property that for every tree $t \in T_\Sigma$ with $p = \widetilde{\mu}(t)$ it holds that $r = \widetilde{\mu}(C'[t])$, then the state r is computable. Let us denote this last statement by (†). Moreover, by the above equivalences it is decidable whether there exists a state $r \in Q$ such that $r = \widetilde{\mu}(C'[t])$

for every tree $t \in T_\Sigma$ with $p = \widetilde{\mu}(t)$. Consequently it is decidability whether $\widetilde{\mu}(t_j) \in Q$ for every index $j \in [k] \setminus \{i\}$ and there exists a state $r \in Q$ such that $r = \widetilde{\mu}(C'[t])$ for every tree $t \in T_\Sigma$ with $p = \widetilde{\mu}(t)$. It remains to show that $h_\mu(t_1)_{\widetilde{\mu}(t_1)} \odot \cdots \odot h_\mu(t_{i-1})_{\widetilde{\mu}(t_{i-1})} \odot a_M(p, r, C')$ is computable. By induction hypothesis $a_M(p, r, C')$ is computable and, since $\text{height}(C) \leq \text{card}(Q) + 1$ and thus $\text{height}(t_j) \leq \text{card}(Q)$ for every index $j \in [k] \setminus \{i\}$, also $h_\mu(t_j)$ and $\widetilde{\mu}(t_j)$ are computable. Hence $h_\mu(t_j)_{\widetilde{\mu}(t_j)}$ and $h_\mu(t_1)_{\widetilde{\mu}(t_1)} \odot \cdots \odot h_\mu(t_{i-1})_{\widetilde{\mu}(t_{i-1})} \odot a_M(p, r, C')$ are computable, which completes the proof. □

Corollary 6.5 (Constant-on-its-support problem). *It is decidable whether the tree series S is constant on its support.*

Proof. Conditions (CPS1) and (CPS2) of Lemma 6.3 are decidable, since the sets B_1 and B_2 are computable by Lemma 6.4. It remains to find the appropriate semiring element $d \in A$, which we define according to Property (CPS2) as follows:

- If $\text{card}(B_2 \setminus \{\mathbf{0}\}) = 0$, then $d \in A$ can be arbitrarily chosen.
- If $\text{card}(B_2 \setminus \{\mathbf{0}\}) = 1$, then we set d such that $\{d\} = B_2 \setminus \{\mathbf{0}\}$.
- If $\text{card}(B_2 \setminus \{\mathbf{0}\}) \geq 2$, then S does not have the desired property.

□

Let us now investigate the Constant problem, i.e., is it decidable whether $S = \widetilde{d}$ for some semiring element $d \in A$? We could prove this by deciding whether $(S, t) = d$ for every input tree $t \in \text{supp}(S)$ (Constant-on-its-support problem) and $\text{supp}(S) = T_\Sigma$. The latter decision problem could be solved as in classical theories (cf. [GS84], Theorem 10.3 of Chapter II, also using the well-known fact that T_Σ is a recognizable tree language) provided that the underlying semiring is zero-divisor free. We would like to avoid this additional assumption. Therefore we present two properties (CP1) and (CP2), which are equivalent to $S = \widetilde{d}$ for a given semiring element $d \in A$ (cf. Lemma 6.6), and then show that this semiring element d can be derived from the Properties (CP1) and (CP2) (cf. Corollary 6.7).

Lemma 6.6. *Let $\mathcal{A}$ be a commutative semiring and $d \in A$ a semiring element. It holds that $S = \widetilde{d}$, if and only if*

(CP1) for every semiring element $b \in B_1$ it holds that $b \odot d = d$ and

(CP2) for every tree $s \in B_2$ it holds that $(S, s) = d$.

Proof. The proof is very similar to the proof of Lemma 6.3 and hence left to the reader. □

Corollary 6.7 (Constant problem). *If $\mathcal{A}$ is a commutative semiring, then it is decidable whether S is a constant tree series.*

Proof. The proof of Corollary 6.5 can be taken over word by word. □

From the results, which we have obtained in this section so far, we now derive several additional decidability results. A straightforward consequence of the decidability of the Constant problems is that the Emptiness problem is decidable.

Corollary 6.8 (Emptiness problem). *If $\mathcal{A}$ is a commutative semiring, then it is decidable whether* $\mathrm{supp}(S) = \emptyset$.

Proof. The claim follows from Lemma 6.3 with $d = 0$. □

Another interesting class of tree series are the boolean tree series. Is it decidable whether a recognizable tree series is boolean? The answer is yes, if the underlying semiring is commutative. The decision procedure uses the fact, that a tree series is boolean, if and only if $(S,t) = \mathbf{1}$ for every input tree $t \in \mathrm{supp}(S)$. We thereby reduce the decidability problem of S being boolean to the Constant-on-its-support problem.

Corollary 6.9 (Boolean problem). *If $\mathcal{A}$ is a commutative semiring, then it is decidable whether S is a boolean tree series.*

Proof. The claim follows from Lemma 6.3 with $d = 1$. □

Let us conclude this section by proving that the Finiteness problem is decidable provided that the underlying semiring is commutative and zero-divisor free. From the pumping lemma (cf. Corollary 5.8) and zero-divisor freeness we deduce that, if there exists a tree $t \in \mathrm{supp}(S)$ of height $\geq \mathrm{card}(Q) + 1$, then $t = C'[C[s]]$ can be decomposed into contexts $C, C' \in C_\Sigma(X_1)$ and a tree $s \in T_\Sigma$ such that pumping the context C produces infinitely many trees $C'[C^n[s]] \in \mathrm{supp}(S)$, $n \in \mathbb{N}$. Hence, $\mathrm{supp}(S)$ is finite, if and only if every tree $t \in \mathrm{supp}(S)$ is of height $\leq \mathrm{card}(Q)$. Let us therefore consider the tree series S', which is defined for every input tree $t \in T_\Sigma$ by

$$(S',t) = \begin{cases} (S,t) & \text{, if height}(t) \geq \mathrm{card}(Q) + 1, \\ \mathbf{0} & \text{, otherwise.} \end{cases}$$

Thus S has finite support, if and only if the support of S' is the empty set. In order to apply the decidability of the Emptiness problem it remains to show that the tree series S' is recognizable by a deterministic bu-w-fta. For this purpose we define for every semiring $\mathcal{A}$ and non-negative integer $n \in \mathbb{N}$ the tree series $S_n^{\mathrm{height}} \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$, which is defined for every input tree $t \in T_\Sigma$ by

$$(S_n^{\mathrm{height}},t) = \begin{cases} \mathbf{1} & \text{, if height}(t) \geq n+1, \\ \mathbf{0} & \text{, otherwise.} \end{cases}$$

We observe that $S' = S \odot S_{\mathrm{card}(Q)}^{\mathrm{height}}$, where $\odot$ denotes the Hadamard product of the tree series S and $S_{\mathrm{card}(Q)}^{\mathrm{height}}$. By Corollary 3.9(ii) it remains to show that the tree series $S_{\mathrm{card}(Q)}^{\mathrm{height}}$ is recognized by some deterministic bu-w-fta in order to prove that S' is accepted by a deterministic bu-w-fta.

Lemma 6.10. *Let $\mathcal{A}$ be a semiring and $n \in \mathbb{N}$ a non-negative integer. The tree series S_n^{height} is recognized by the deterministic bu-w-fta $M_n^{\text{height}} = (Q', \Sigma, \nu', \mathcal{A}, \mu')$, which is defined by $Q' = \{1, \ldots, n, n+1\}$, $\nu'(n+1) = \mathbf{1}$, $\nu(q') = \mathbf{0}$ for every state $q' \in Q' \setminus \{n+1\}$, and, for every non-negative integer $k \in \mathbb{N}$, input symbols $\alpha \in \Sigma^{(0)}$ and $\sigma \in \Sigma^{(k)}$, and states $q'_1, \ldots, q'_k, q' \in Q'$,*

$$\mu'_0(\alpha)_{(),q'} = \begin{cases} \mathbf{1} & \text{, if } q' = 1, \\ \mathbf{0} & \text{, otherwise,} \end{cases}$$

$$\mu'_k(\sigma)_{(q'_1,\ldots,q'_k),q'} = \begin{cases} \mathbf{1} & \text{, if } (q' = \max\{q'_i \mid i \in [k]\} + 1) \text{ or} \\ & \quad (q' = n+1 \text{ and } (\exists i \in [k]) : q'_i = n+1), \\ \mathbf{0} & \text{, otherwise.} \end{cases}$$

Proof (Sketch). First we observe that M' is a deterministic bu-w-fta. A straightforward inductive proof shows that

(i) for every input tree $t \in T_\Sigma$ of height $\leq n$ it holds that $h_{\mu'}(t)_{\text{height}(t)} = \mathbf{1}$ and $h_{\mu'}(t)_{q'} = \mathbf{0}$ for every state $q' \in Q' \setminus \{\text{height}(t)\}$.

(ii) for every input tree $t \in T_\Sigma$ of height $> n$ it holds that $h_{\mu'}(t)_{n+1} = \mathbf{1}$ and $h_{\mu'}(t)_{q'} = \mathbf{0}$ for every state $q' \in Q' \setminus \{n+1\}$.

Hence, $(S_{M'}, t) = \sum_{q' \in Q'} h_{\mu'}(t)_{q'} \odot \nu'(q') = h_{\mu'}(t)_{n+1} = (S_n^{\text{height}}, t)$ for every input tree $t \in T_\Sigma$ and thus the tree series S_n^{height} is recognized by the deterministic bu-w-fta M'. □

Theorem 6.11 (Finiteness problem). *If $\mathcal{A}$ is a zero-divisor free and commutative semiring, then it is decidable whether* $\text{supp}(S)$ *is a finite set.*

Proof. First we show that S has finite support, if and only if every input tree $t \in \text{supp}(S)$ is of height $\leq \text{card}(Q)$. We denote this statement by (*). Let us prove the non-trivial implication of (*) by contradiction, i.e., let $\text{supp}(S)$ be a finite set and assume that there exists an input tree $t \in \text{supp}(S)$ with $\text{height}(t) \geq \text{card}(Q)+1$. By Corollary 5.8 there exists a decomposition $t = C'[C[s]]$ of t into contexts $C', C \in C_\Sigma(X_1)$ and a tree $s \in T_\Sigma$ such that C is not the trivial context x_1 and there exist semiring elements $a, a', b, b', c \in A$ with $(S, C'[C^n[s]]) = a' \odot a^n \odot c \odot b^n \odot b'$ for every non-negative integer $n \in \mathbb{N}$. In particular, $a \neq \mathbf{0}$, $b \neq \mathbf{0}$, and $(S, C'[C^n[s]]) = a^{n-1} \odot b^{n-1} \odot (S, t)$ for every positive integer $n \in \mathbb{N}_+$. By zero-divisor freeness and $t \in \text{supp}(S)$ we have $(S, C'[C^n[s]]) \neq \mathbf{0}$ and thus $C'[C^n[s]] \in \text{supp}(S)$ for every positive integer $n \in \mathbb{N}_+$, which either contradicts to $C \neq x_1$ or tol $\text{supp}(S)$ is a finite set.

By (*) the statement $\text{supp}(S)$ is a finite set is equivalent to $S' = \widetilde{\mathbf{0}}$, i.e., $\text{supp}(S') = \emptyset$. Let us denote this statement by (†). In order to apply the decidability result for the Emptiness problem (cf. Corollary 6.8, it remains to show that S' is recognized by a deterministic bu-w-fta. Since $S' = S \odot S_{\text{card}(Q)}^{\text{height}}$, we deduce from Lemmata 3.8 and 6.10 that the bu-w-fta $M \times M_{\text{card}(Q)}^{\text{height}}$ accepts S'. Moreover, since M and

$M^{\mathsf{height}}_{\mathsf{card}(Q)}$ are deterministic bu-w-fta, also $M \times M^{\mathsf{height}}_{\mathsf{card}(Q)}$ is deterministic. Hence Corollary 6.8 is applicable and thus, by (†), it is decidable whether $\mathrm{supp}(S)$ is a finite set. □

Acknowledgment

The author wishes to thank the anonymous referees for helpful comments.

References

[AB87] A. Alexandrakis and S. Bozapalidis. Weighted grammars and Kleene´s theorem. *Information Processing Letters*, 24(1):1–4, January 1987.

[BA89] S. Bozapalidis and A. Alexandrakis. Representation Matricielles de Series d'Arbre Reconnaissables. *Theoretical Informatics and Applications*, 23(4):449–459, 1989.

[BGW00] A. L. Buchsbaum, R. Giancarlo, and J. R. Westbrook. On the determinization of weighted finite automata. *SIAM Journal on Computing*, 30(5):1502–1531, 2000.

[Bor03] B. Borchardt. The Myhill-Nerode Theorem for Recognizable Tree Series. In *7th International Conference on Developments in Language Theory, DLT'03, Szeged, Hungary, July 7-11, 2003, Proceedings*, volume 2710 of *Lecture Notes in Computer Science*, pages 146–158. Springer-Verlag, July 2003.

[Boz91] S. Bozapalidis. Effective construction of the syntactic algebra of a recognizable series on trees. *Acta Informatica*, 28:351–363, 1991.

[Boz94] S. Bozapalidis. Representable Tree Series. *Fundamenta Informaticae*, 21:367–389, 1994.

[Boz97] S. Bozapalidis. Positive Tree Representations and Applications to Tree Automata. *Information and Control*, 139(2):130–153, 1997.

[Boz99] S. Bozapalidis. Equational Elements in Additive Algebras. *Theory of Computing Systems*, 32(1):1–33, 1999.

[Boz01] S. Bozapalidis. Context-Free Series on Trees. *Information and Computation*, 169:186–229, 2001.

[BR82] J. Berstel and C. Reutenauer. Recognizable formal power series on trees. *Theoretical Computer Science*, 18(2):115–148, 1982.

[BR88] J. Berstel and Ch. Reutenauer. *Rational Series and Their Languages*, volume 12 of *EATCS-Monographs*. Springer-Verlag, 1988.

[BV03] B. Borchardt and H. Vogler. Determinization of Finite State Weighted Tree Automata. *Journal of Automata, Languages and Combinatorics*, 8(3):417–463, 2003.

[DK03] M. Droste and D. Kuske. Skew and infinitary formal power series. In *ICALP 2003*, volume 2719 of *LNCS*, pages 426–438. Springer-Verlag, 2003.

[DPV03] M. Droste, C. Pech, and H. Vogler. A Kleene theorem for weighted tree automata. *Theory of Computing Systems*, 2003. to appear.

[DV03] M. Droste and H. Vogler, editors. *Workshop on Weighted Automata: Theory and Applications, Dresden, Germany, March 4-8, 2002, Selected Papers*, volume 8(2) of *Journal of Automata, Languages and Combinatorics*. Otto-von-Guericke-Universität Magdeburg, 2003.

[EFV02] J. Engelfriet, Z. Fülöp, and H. Vogler. Bottom-up and top-down tree series transformations. *J. Automata, Languages and Combinatorics*, 7:11–70, 2002.

[Eil74] S. Eilenberg. *Automata, Languages, and Machines – Volume A*, volume 59 of *Pure and Applied Mathematics*. Academic Press, 1974.

[ÉK03] Z. Ésik and W. Kuich. Formal tree series. In Droste and Vogler [DV03], pages 219–285.

[Eng75] J. Engelfriet. Bottom-up and top-down tree transformations - a comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975.

[FSW94] C. Ferdinand, H. Seidl, and R. Wilhelm. Tree automata for code selection. *Acta Informatica*, 31(8):741–760, 1994.

[FV03] Z. Fülöp and H. Vogler. Tree series transformations that respect copying. *Theory of Computing Systems*, 36(3):247–293, 2003.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[GS97] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer-Verlag, 1997.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison–Wesley, 1979.

[HW98] U. Hebisch and H.J. Weinert. *Semirings - Algebraic Theory and Applications in Computer Science*. World Scientific, Singapore, 1998.

[KS86] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1986.

[Kui97a] W. Kuich. Formal Power Series over Trees. In S. Bozapalidis, editor, *3rd International Conference on Developments in Language Theory, DLT 1997, Thessaloniki, Greece, July 20-23, 1997, Proceedings*, pages 61–101. Aristotle University of Thessaloniki, 1997.

[Kui97b] W. Kuich. Semirings and formal power series: Their relevance to formal languages and automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 1*, chapter 9, pages 609–677. Springer-Verlag, 1997.

[Kui99] W. Kuich. Tree transducers and formal tree series. *Acta Cybernetica*, 14:135–149, 1999.

[Moh97] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311 (1–42), 1997.

[Sch61] M.P. Schützenberger. On the definition of a family of automata. *Inf. and Control*, 4:245–270, 1961.

[Sei94] H. Seidl. Finite tree automata with cost functions. *Theoretical Computer Science*, 126(1):113–142, 1994.

Received October, 2003

Acta Cybernetica 16 (2004) 545–566.

An addition to the methods of test determination for fault detection in combinational circuits

Ljubomir Cvetković*

Abstract

We propose a procedure for determining fault detection tests for single and multiple fault in combinational circuits. The stuck-at-fault model is used. By the proposed procedure all test vectors for single and multiple stuck-at-fault in combinational circuit are determined. The path sensitization method is used in the test signal propagation while test signals are defined on a four element set. The procedure can also be applied to the fault detection in programmable logic devices. We consider two-level combinational circuits which are realized by the PAL architecture and we propose a procedure for determining a test set which detects all single stuck-at-faults. As a mathematical tool, the cube theory is used.

Keywords: combinational circuit, programmable logic devices, single fault, multiple fault, fault detection, test cube

1 Introduction

In literature many approaches for fault detection in combinational circuits can be found (see, for example, [1, 13, 14]).

Test generation problem can be considered viewed as a finite space search problem of finding appropriate logic value assignments to the primary inputs such that the given fault is detected. Since the size of the search space is exponential in the number of inputs, the test generation problem is proven to be NP-complete [9].

In the stuck-at-fault model it is assumed, that a faulty line of the combinational circuit is permanently set to either the value of 0 or 1. The corresponding faults are called stuck-at-0 and stuck-at-1 respectively. Combinational circuit lines are understood to be all connections along which logic signals move. The fault 0 (or 1) at the line a is denoted by $a/0$ (or $a/1$).

The procedure for the single stuck-at-fault detection in a combinational circuit is based on the following two conditions. First, one should generate on the faulty line a test signal whose value is opposite to the fault signal. Secondly, one should enable the test signal propagation to the output line of the combinational circuit,

*Ljubomir Cvetković, Trg Žitna Pijaca 3, 22000 Sremska Mitrovica, Serbia and Montenegro, JP PTT "SERBIA"-HOLDING, Belgrade, Serbia and Montenegro. E-mail: ljubomir@ptt.yu

which means that the path from the faulty line to the output line must be active. Such an approach to fault detection is known as the path sensitization method. The path sensitization method is presented for the first time by Armstrong [5], where it is applied to the single fault detection. Afterwards some new procedures for the multiple fault detection are developed.

The D-algorithm is one of the first algorithms for automatic generation of test vectors for single stuck-at-faults in combinational circuits. This algorithm has been developed by Roth [17]. Test signals take five algebraic values: $0, 1, X, \bar{D}$ and D. Logical zero and one are denoted by 0 and 1, respectively, while X belongs to the set $\{0, 1, \bar{D}, D\}$, where D is equal to 1 and 0 in fault free and faulty circuit, respectively.

One has quickly realized that the D-algorithm should be improved. The reduction of the number of paths is achieved by means of the so called 9-V algorithm, whose structure is closest to the D-algorithm [8].

In construction of the algorithms for automatic test generation for combinational circuits the notions of D-boundary and P-boundary are defined [1]. The D-boundary is the set of logic gates whose output lines are in state X, while at least one test signal with values D and $\bar{D}$ appears on the input lines of these gates. It is necessary to choose one of the logic gates from the D-boundary, together with determination of the signals on its input lines, in such a way that output line gets the value D or $\bar{D}$, in order to direct the test signal propagation towards the primary output. The P-boundary is the set of logic gates, in which the state of the output line is fully determined, while the state of values on the input lines is undetermined.

A more efficient decision making process was proposed almost 20 years ago in the PODEM algorithm [10]. PODEM only allows assignment of values at primary inputs (PIs) and these values are then propagated towards internal lines using implications. After the initial assignment of values on some primary inputs, and implication of these values, the process continues by assigning additional PI values and checking if the fault effect has been propagated to the primary outputs. For the combinational circuits, in which the number of paths is large, we have to enumerate and label all the paths. The algorithms are based on the search on graphs and are the subject of research [16].

Generally speaking, a lot of procedures for testing combinational circuits have been developed. Most of them are designed for restricted type circuits. For fan-out free combinational circuits the test set which detects all single stuck-at-faults, also detects all multiple stuck-at-faults [12].

It is well known that in a two-level irredundant combinational circuit every set of test vectors which detects all single stuck-faults detects at the same time all multiple stuck-at-faults [7], p. 65.

The AND/OR structure of a two-level circuit corresponds naturally to sum of products structure of disjunctive-normal-form. All single and multiple stuck-at-faults are detectable in such circuits if all single faults at input and output lines of the AND gates are detectable [11].

In the literature more subtle and specific types of faults have been derived for PLAs.

Since PLAs can be used to implement any two-level logic expressions, we can treat the PLA as a two-level logic gate implementation and model the typical stuck-at-faults at input, input inverters, product lines and output lines.

A fault model, according to which incorrect logical connections in AND and OR arrays are tested, is proposed in [18]. Since AND gate inputs and OR gate inputs can be either improperly connected or disconnected, modeled faults can be conveniently divided into four classes. First, if an input literal is disconnected from AND gate, this causes the implicant to "grow" since the implicant becomes independent of some input variable. This growth of an implicant can be seen quite easily on a Karnaugh map. These faults are called growth faults. The set of all single growth faults are denoted as G. If an AND gate becomes disconnected from an OR gate, this causes an implicant to disappear from the map of the function. Hence, this set of single faults is the set of disappearance faults D. If an input literal becomes incorrectly connected to an AND gate, then the implicant "shrinks". These faults form the set of shrinkage faults S. Finally, if an AND gate becomes incorrectly connected to an OR gate, than an implicant appears in the map of the affected functions. Hence, these are called appearance faults A.

There could be also bridging faults in the PLAs. Especially the bridging faults could easily happen between the adjacent lines due to the regular layout style of PLAs. The effects of the bridging could be AND or OR depend on the technology on which PLA realizes.

In PLA and PAL, the test sets, which detects cross points faults, detect at the same time a great number of stuck-at-faults and short faults. Cross point faults are important in this devices and therefore several procedures for detecting these faults have been developed (see, for example [3, 4]).

The paper [19] presents a method for obtaining a minimal set of test configurations and their associated set of test patterns that completely tests re-programmable Programmable Logic Arrays (PLAs) including EEPROM, UV-EPROM, and SRAM based re-programmable PLAs typically found in Complex Programmable Logic Devices (CPLDs). The resultant set of test configurations and vectors detect all single and multiple stuck-at-faults (including line and transistor faults) as well as all bridging faults in the PLA.

Bridge defects also cause less obvious effects that can introduce further modelling complications. If a bridge defect creates a feedback loop, a formerly stable combinational circuit may take on oscillatory or sequential properties. Also, when intermediate voltage levels occur, downstream logic gates with varying input voltage thresholds can interpret the same voltage level differently. This is known as the *Byzantine General's Problem* and can significantly complicate bridge defect modelling issues [2].

The problem of determining a minimal number of control inputs for converting a programmable logic array (PLA) with undetectable faults to crosspoint-irredundant PLA for testing has been formulated as a nonstandard set covering problem. By representing subsets of sets as cubes, this problem has been reformulated as familiar problems. It is noted that this result has significance because a crosspoint-irredundant PLA can be converted to a completely testable PLA in a straightfor-

ward fashion, thus achieving very good fault coverage and easy testability [15].

A system of Boolean functions can be realized by blocks arranged in several levels, where each block represents a PAL with a small number of inputs and outputs. Moreover, multi-level circuits are harder tested than two-level circuits [6].

In this paper we propose the stuck-at-fault model for PLA and assume that the programmable elements, where the faults appear, get the signal values stuck-at-0 or stuck-at-1. The acceptance of the proposed model is motivated by the fact that the faults in real PLA appear in programmable elements. Therefore, we accept the following approximation of defects: PLA programmable points, where faults appear, get the signal values permanent logic 1 or permanent logic 0. This approximation is the main strength of the proposed model. It is well known that the quality of defect abstraction has an influence on the relevance of the fault model. On the other hand, the defect abstraction by faults has an influence on the testing methodology. In our case it is appropriate to use the path sensitization method as a testing method and to introduce a four element signal value set. We shall show in the sequel that the fault types, appearing in the above mentioned standard fault models for PLA, can be detected by the proposed procedure as well, thus proving the relevance and sufficiency of our model. Details on the fault models comparison are given in section 5.

2 Test signal propagation

The values of test signals are denoted by D and C. Test signal D has value 1 in a fault-free combinational circuit, and it has value 0 in a faulty combinational circuit. Test signal C has value 0 in a fault-free combinational circuit, and it has value 1 in a faulty combinational circuit. Test signals have values from the set $\{0, 1, C, D\}$ where $C, D \in \{0, 1\}$. It should be noted that values of C and D are not determined at the beginning of the fault detection procedure; they are concretized during the execution of the procedure. The set $\{0, 1, C, D\}$ and operations $+, \bullet$ and $\bar{}$ represent a Boolean algebra, if the operations are defined by the following tables

Table 1: Operation $+$

$+$	0	1	C	D
0	0	1	C	D
1	1	1	1	1
C	C	1	C	1
D	D	1	1	D

Table 2: Operation $\bullet$

$\bullet$	0	1	C	D
0	0	0	0	0
1	0	1	C	D
C	0	C	C	0
D	0	D	0	D

The intersection operation ϕ of the set $\{0, 1, X\}$ is defined by Table 4. Symbol $\emptyset$ in Table 4 means that operation ϕ is not defined.

Of course, using a four element set in constructing test vectors is well- known [7], p.44. The inputs can take the four values $\{0, 1, \bar{D}, D\}$ where the symbols D

Table 3: Operation $^-$

$^-$	0	1	C	D
	1	0	D	C

Table 4: Operation ϕ

ϕ	0	1	X
0	0	$\emptyset$	0
1	$\emptyset$	1	1
X	0	1	X

and $\bar{D}$ are used to represent respectively a signal which has the value 1 and 0 in a well functioning circuit and 0 and 1 in a faulty circuit.

Consider now a logic gate G with n input lines and output line v (Fig.1). Gate G can be OR, NOR, AND or NAND. Suppose that test signals of value D or C appear on input lines $u_1, u_2, \ldots, u_p$ of gate G while propagation signal values 0 or 1 appear on the remaining lines $z_1, z_2, \ldots, z_q$ $(p+q=n)$.

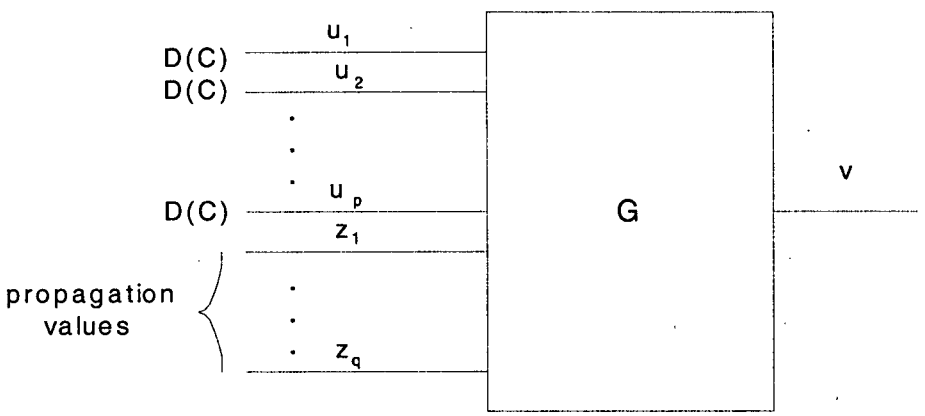


Figure 1: Test signals have the same value

The test signal propagation to the output line v can occur in the case when test signals of the same value (D or C) appear on input lines $u_1, u_2, \ldots, u_p$ while signal values on the remaining lines $z_1, z_2, \ldots, z_q$ must have value 0 for gates OR and NOR, and value 1 for gates AND and NAND.

Let us analyze conditions under which test signals are propagated from input lines to the output line of gates OR, NOR, AND and NAND. Logical operation "$\wedge$" is used in logical relations which describe the test signal propagation.

The following logical relations hold:
for OR gate

$$(u_1 = D) \wedge \cdots \wedge (u_p = D) \wedge (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = D), \qquad (1)$$

$$(u_1 = C) \wedge \cdots \wedge (u_p = C) \wedge (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = C), \qquad (2)$$

for NOR gate

$$(u_1 = D) \wedge \cdots \wedge (u_p = D) \wedge (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = C), \tag{3}$$

$$(u_1 = C) \wedge \cdots \wedge (u_p = C) \wedge (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = D), \tag{4}$$

for AND gate

$$(u_1 = D) \wedge \cdots \wedge (u_p = D) \wedge (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = D), \tag{5}$$

$$(u_1 = C) \wedge \cdots \wedge (u_p = C) \wedge (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = C), \tag{6}$$

for NAND gate.

$$(u_1 = D) \wedge \cdots \wedge (u_p = D) \wedge (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = C), \tag{7}$$

$$(u_1 = C) \wedge \cdots \wedge (u_p = C) \wedge (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = D). \tag{8}$$

In relations (1)-(8) test signal values D and C appear p times each. Test signal on the output line v can be obtained also by setting some test signals on input lines $u_1, u_2, \ldots, u_p$ to propagation signal values. This can be done with each k $(1 \leq k \leq p-1)$ element subset of the set of input lines $\{u_1, u_2, \ldots, u_p\}$. Hence the total number of logical relations is

$$\binom{p}{1} + \binom{p}{2} + \cdots + \binom{p}{p-1} = 2^p - 2 \tag{9}$$

Consider the situation presented in Fig. 2. Test signal value D appears on r input lines of the gate G while value C appears on s input lines. Lines $z_1, z_2, \ldots, z_q$ have propagation signal values for gate G. In this case no one of the relations (1)-(8) can be applied since $D \cdot C = 0$.

Suppose that gate G on Fig.2 is an OR gate.

Test signal of value D is propagated to the output line v according to the relation

$$(u_1 = D) \wedge \cdots \wedge (u_r = D) \ \wedge \ (u_{r+1} = 0) \wedge \cdots \wedge (u_{r+s} = 0) \\ \wedge \ (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = D). \tag{10}$$

Test signal value C appearing s times in relation (10) is replaced by 0. Hence, test signal values D appear r times in relation (10), while the remaining $s+q$ $(r+s+q=n)$ input lines of the gate OR must have values of propagation signals. For the propagation of the test signal of value D to the output line one can write $2^r - 2$ relations.

Test signal of value C is propagated to the output line v according to the relation

$$(u_1 = 0) \wedge \cdots \wedge (u_r = 0) \ \wedge \ (u_{r+1} = C) \wedge \cdots \wedge (u_{r+s} = C) \\ \wedge \ (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = C). \tag{11}$$

Analogously to the above analysis, for the propagation of the test signal of value C to the output line v one can write $2^s - 2$ relations.

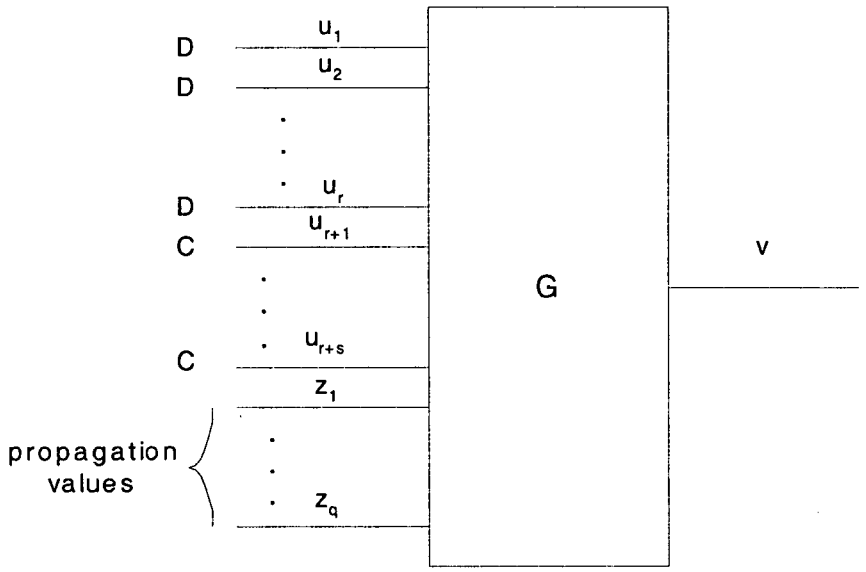


Figure 2: Test signals with distinct values

The following logical relations hold:
for NOR gate

$$(u_1 = D) \wedge \cdots \wedge (u_r = D) \ \wedge \ (u_{r+1} = 0) \wedge \cdots \wedge (u_{r+s} = 0) \\ \wedge \ (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = C), \qquad (12)$$

$$(u_1 = 0) \wedge \cdots \wedge (u_r = 0) \ \wedge \ (u_{r+1} = C) \wedge \cdots \wedge (u_{r+s} = C) \\ \wedge \ (z_1 = 0) \wedge \cdots \wedge (z_q = 0) \rightarrow (v = D), \qquad (13)$$

for AND gate

$$(u_1 = D) \wedge \cdots \wedge (u_r = D) \ \wedge \ (u_{r+1} = 1) \wedge \cdots \wedge (u_{r+s} = 1) \\ \wedge \ (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = D), \qquad (14)$$

$$(u_1 = 1) \wedge \cdots \wedge (u_r = 1) \ \wedge \ (u_{r+1} = C) \wedge \cdots \wedge (u_{r+s} = C) \\ \wedge \ (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = C), \qquad (15)$$

for NAND gate

$$(u_1 = D) \wedge \cdots \wedge (u_r = D) \ \wedge \ (u_{r+1} = 1) \wedge \cdots \wedge (u_{r+s} = 1) \\ \wedge \ (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = C), \qquad (16)$$

$$(u_1 = 1) \wedge \cdots \wedge (u_r = 1) \ \wedge \ (u_{r+1} = C) \wedge \cdots \wedge (u_{r+s} = C) \\ \wedge \ (z_1 = 1) \wedge \cdots \wedge (z_q = 1) \rightarrow (v = D). \qquad (17)$$

3 Fault detection

In the sequel we propose a procedure for the single fault detection for fan-out free combinational circuits.

Let a unique path $(h, e_1, e_2, \ldots, e_n, y)$ in combinational circuit go from the faulty line h to the output line y. The first logical relation defines the conditions of the test signal propagation from the line h of the distance d (from the output line) to the line e_1 that has a distance $d-1$. Line e_1 is the logic gate output line whose input line is h. The second logical relation defines the conditions for test signal propagation from the line e_1 of the distance $d-1$, to the line e_2 that has a distance $d-2$. The process of logical relations writing continues as far as the last logical relation is found. It defines conditions for the test signal propagation from the line e_n of the distance 1, to the output line y in combinational circuit. Uniting the d logical relations into a single logical relation is obvious. The test signal value from the right hand side of the first logical relation is replaced into the second relation. Then the test signal value from the right hand side of the second logical relation is replaced into the third relation and the process continues.

Obviously, when writing logical relations we go only forward, unsuccessful steps being excluded in this case. For a single stuck-at-fault, we surely determine all test vectors. At the and of the procedure one obtains a logical relation in which signals on input lines appear. Based on the test signal values on the input lines one can determine sets of test cubes and on the basis of them we determines the detecting test set.

Example 1. Determine all test vectors detecting a single fault $h/1$ in the combinational circuit in Fig. 3.

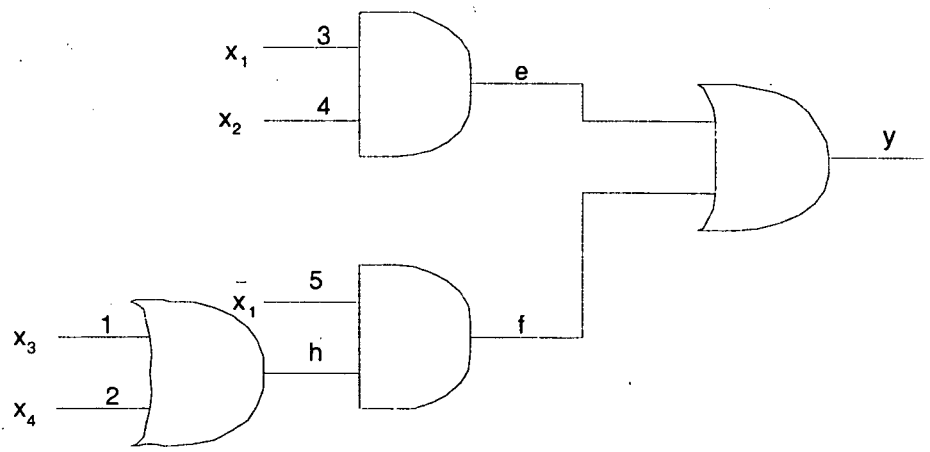


Figure 3: A single fault in a combinational circuit

Table 5: Auxiliary data

$h=0$	$XX00$
5=1	$0XXX$
$e=0$	$0XXX$
	$X0XX$

The unique path from the faulty line h to the output of the circuit is (h, f, y). The signal having the value C should be generated on the faulty line h. Logical relations defining conditions for test signal propagation from faulty line to the output of the combinational circuit are

$$(h = C) \wedge (5 = 1) \rightarrow (f = C), \tag{18}$$

$$(f = C) \wedge (e = 0) \rightarrow (y = C). \tag{19}$$

On the basis of relations (18) and (19) the next relation is derived

$$(h = C) \wedge (5 = 1) \wedge (e = 0) \rightarrow (y = C). \tag{20}$$

Line covers for h, 5 and e are shown in Table 5.
Test cubes detecting the fault $h/1$ are obtained on the basis of Table 5

$$\{XX00\}\phi\{0XXX\}\phi\{0XXX, X0XX\} = \{0X00, 0000\}.$$

Finally, we get the set of all test vectors for the given single fault: $\{0000, 0100\}$.

The described procedure for detecting single stuck-at-faults can be applied also for detection of multiple stuck-at-fault. Under a multiple fault we understand a fault consisting of single faults. In order to detect a multiple fault, one should generate test signals with values complementary to the fault signals at the faulty lines of the combinational circuit. To describe the test signal propagation we use logical relations described in Section 2.

In the next example a double fault is located on two paths in the combinational circuit, the two paths having some lines in common.

Example 2. Let us detect the double fault $\{1/0, 3/0\}$ in combinational circuit on Fig. 4.

To lines 1 and 3 we associate the test signals, having values opposite to the fault signals. We have relations

$$(1 = D) \wedge (2 = 1) \rightarrow (e = D), \tag{21}$$

$$(3 = D) \wedge (4 = 0) \rightarrow (f = D). \tag{22}$$

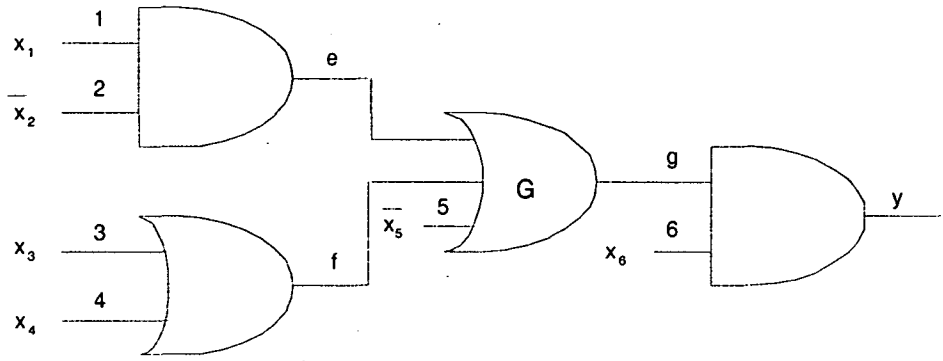


Figure 4: A double fault in a combinational circuit

The propagation of the test signal from input lines e and f of gate G to its output line is described by the relation

$$(e = D) \wedge (f = D) \wedge (5 = 0) \rightarrow (g = D). \tag{23}$$

We write a logical relation which describes the propagation of the test signal from line g to the output line y of the combinational circuit

$$(g = D) \wedge (6 = 1) \rightarrow (y = D). \tag{24}$$

On the basis of relations (23) and (24) we have

$$(e = D) \wedge (f = D) \wedge (5 = 0) \wedge (6 = 1) \rightarrow (y = D). \tag{25}$$

Putting relations (21) and (22) into relation (25) we get

$$(1 = D) \wedge (2 = 1) \wedge (3 = D) \wedge (4 = 0) \wedge (5 = 0) \wedge (6 = 1) \rightarrow (y = D). \tag{26}$$

According to (26) we have

$$\{1XXXXX\}\phi\{X0XXXX\} \ \phi \ \{XX1XXX\}\phi\{XXX0XX\}\phi \\ \phi \ \{XXXX1X\}\phi\{XXXXX1\} = \{101011\}.$$

Test vector 101011 detects the double fault $\{1/0, 3/0\}$.

In redundant combinational circuits we can come across the effect of masking. Masking is a phenomenon which can appear in detecting a multiple fault. If α and β are single faults in a combinational circuit of a general structure, the following situation can occur. The single fault α is detectable (and is detected by a certain test set). However, the fault cannot be detected in the presence of the fault β , i.e. in the case of double fault $\{\alpha, \beta\}$. We say that the fault β masks the fault α.

4 Fault detection in programmable logic devices

The above described procedures for fault detection, where test signals values belong to the set $\{0, 1, C, D\}$, can also be applied to the fault detection in programmable logic devices.

A PLA with n input lines, m internal lines and p output lines is represented in Fig. 5.

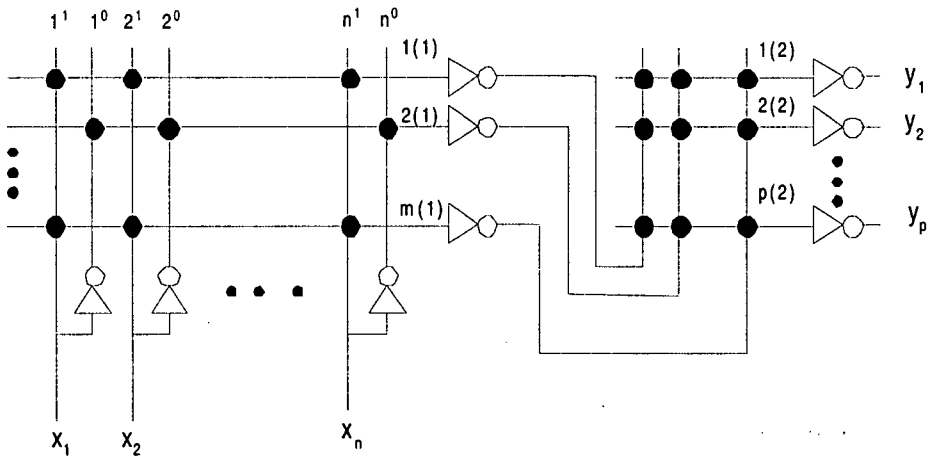


Figure 5: Programmable logic array

Programmable elements are denoted by symbol •. We apply the following way of marking programmable points of PLA:

(i, j^1) - crosspoint of internal line i and a bit line j^1 in AND array $(i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$

(i, j^0) - crosspoint of internal line i and a bit line j^0 in AND array $(i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$

(i, j) - crosspoint between the lines i and j in OR array $(i = 1, 2, \ldots, p,\ j = 1, 2, \ldots, m)$

Within the stuck-at-fault model, PLA programmable points where faults appear, get the signal values stuck-at-0 or stuck-at-1.

The fault of the type of disconnection of a programmable element at point (i, j^1) is denoted by $(i, j^1)/1$ (stuck-at-1 fault), and the fault of the type of a short connection at point (i, j^1) is denoted by $(i, j^1)/0$ (stuck-at-0 fault). Fault marking at the point (i, j^0) is carried out in a similar way.

The fault in the form of disconnection of the programmable element within OR array at the point (i, j) is marked by $(i, j)/1$, and the fault in the form of a short connection at the point (i, j) is marked by $(i, j)/0$.

Example 3. In the programmable logic array presented in Fig. 6, detect multiple fault $\{(1,1)/0,(1,2)/0,(1,3)/0,(1,4)/0)\}$.

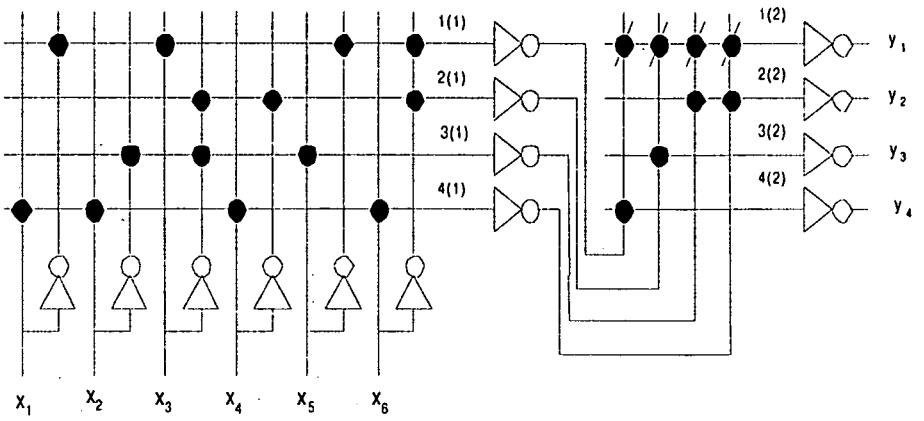


Figure 6: A multiple fault in programmable logic array

It is necessary to generate test signals of value D at points (1,1), (1,2), (1,3) and (1,4). The following logical relation describes the propagation of the test signals to the output line 1(2)

$$(1,1) = D \wedge (1,2) = D \wedge (1,3) = D \wedge (1,4) = D \rightarrow 1(2) = D. \qquad (27)$$

Obviously, test signals of value D exist on the input lines of the OR array on the basis of relation (27). Tracing back, these test signals are attributed to the output lines of the AND array, i.e. $1(1) = C$, $2(1) = C$, $3(1) = C$ and $4(1) = C$. Signals of value C or 1 are attributed to the programmable points of the AND array, where at no pair of bit lines identical signals appear. Such an arrangement of test signals is presented in Fig. 7.

On the basis of Fig. 7 we have

$$Q_1 = \{1X1X10\}, Q_2 = \{XX11X0\}, Q_3 = \{X01X1X\}, Q_4 = \{10X1X0\},$$

which implies

$$Q_1 \phi Q_2 \phi Q_3 \phi Q_4 = \{101110\}.$$

Test vector 101110 detects the given multiple fault.

In order to obtain the remaining test vectors it is necessary to construct other test signals arrangements in the programmable points of the AND array.

Of course, for a PLA with a great number of input lines it is necessary to perform a special analysis (using combinatorics) for constructing optimal algorithms assigning test signals to the programmable points of an AND array. Obviously, one

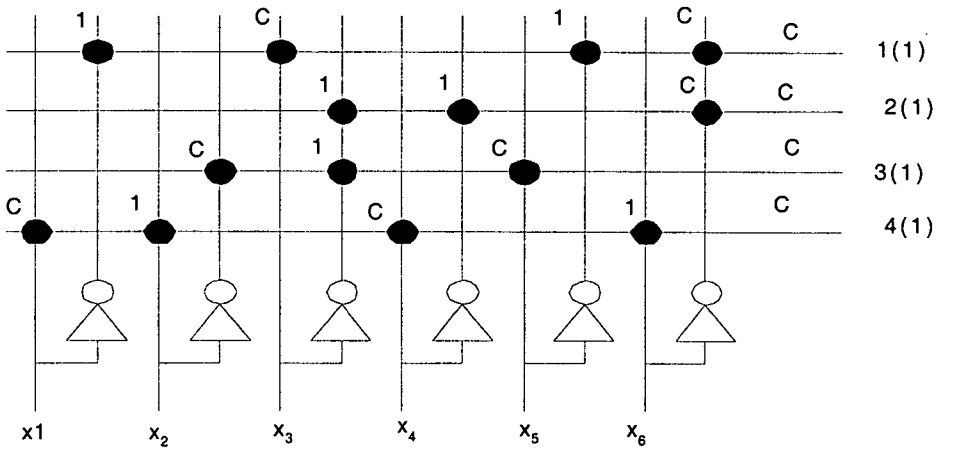


Figure 7: A test signals arrangement in the AND array

or several test vectors correspond to a given test signals arrangement in the AND array. In any case, the problem described is very complex and requires further research.

For Boolean function, a primary and irredundant cover is represented by a two-level combinational circuit for which a test set for all single faults can be found. Obviously, each minimal disjunctive form is irredundant. It is well-known, that a test set, which detects all single stuck-at-faults in irredundant two-level combinational circuits, detects also all multiple stuck-at-faults.

In this paper we assume that the PAL architecture (Fig.8) is used for the realization of two-level irredundant combinational circuits. The output lines of the decoder, in which the minterms of the function y are realized, are connected to the input lines of the OR gate.

Function y is irredundant in the sense that no one of its minterms can be deleted without changing its value. It is well known that irredundant logic can be tested.

In what follows we propose a procedure for constructing a test set which detects all single stuck-at-faults for the PAL architecture in Fig. 8.

In order to construct a procedure for finding the test set for all single stuck-at-faults, we represent the PAL architecture of Fig. 8 by the equivalent two-level combinational circuit with AND/OR gates (Fig. 9).

We apply signals $S_1 = \{100\ldots0, 010\ldots0, \ldots, 000\ldots1\}$ and $S_0 = \{000\ldots0\}$ to input lines of the OR gate in the combinational circuit of Fig.9. Test Signals from $S_1 \cup S_0$ detect $2q+2$ single faults of the OR gate if this gate is considered separately.

Definition 4. *A fault α dominates a fault β if for the set of tests T_α which detect α, and the set of tests T_β which detect β, we have $T_\beta \subset T_\alpha$.*

Let $T = \{X\}$ be the set of input vectors $X = x_1x_2\ldots x_n$ which act on input

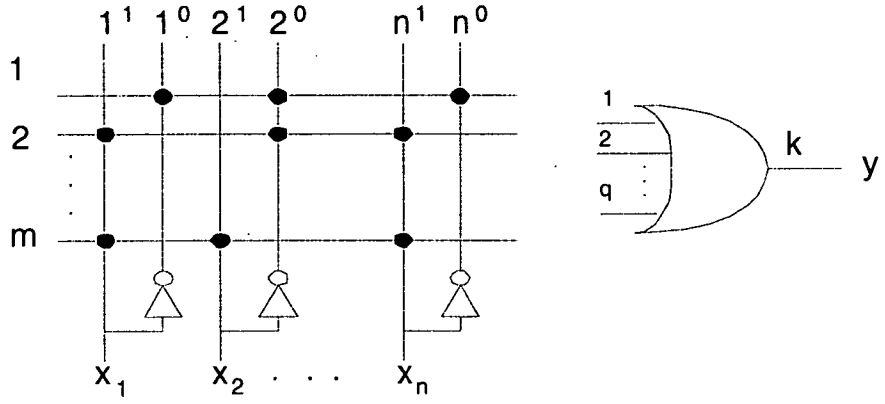


Figure 8: Implementation of Boolean function

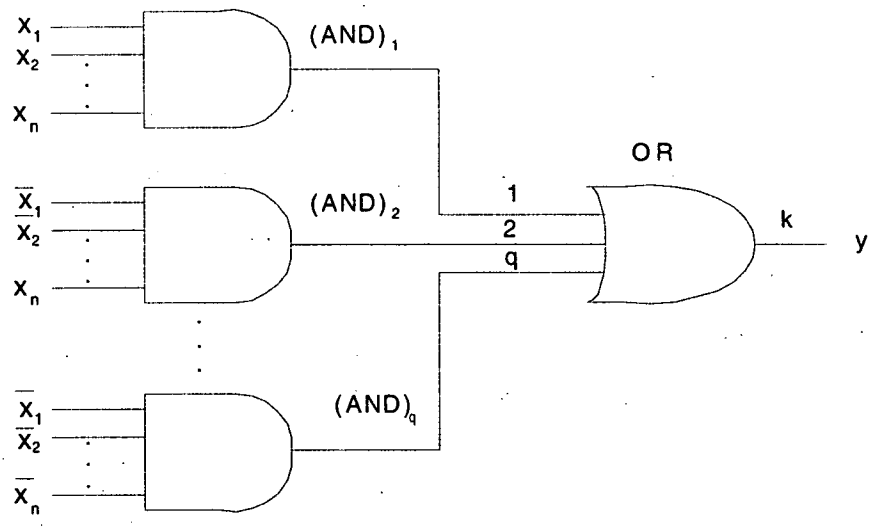


Figure 9: Equivalent combinational circuit

lines of the combinational circuit, i.e. before possible negations, with the property of producing a vector from $S = S_1 \cup S_0$ at the input lines of the OR gate.

Theorem 5. *The set T detects all single stuck-at-faults.*

Proof. Signal values at the output line k in the combinational circuit depend only on the signal values from the set $S_1 \cup S_0$, which act at input lines of the OR gate. The corresponding vectors from T will detect $2q+2$ single stuck-at-faults of the OR

gate. Since signals from $S_1 \cup S_0$ are dominant with respect to signals at the input lines in the combinational circuit, the set T will detect also all single stuck-at-faults of the input lines of the combinational circuit. Hence, the set T detects all single stuck-at-faults in the combinational circuit of Fig. 9. □

A cube which generates signal values 0 or 1 at a given line in the combinational circuit, is called 0-cover or 1-cover respectively.

A cube generates a signal $s \in \{0,1\}$ at any line in combinational circuit which will be denoted by $i = s$. We shall say that the cube satisfies relation $i = s$.

Consider arbitrary lines **i** and **j** in the combinational circuit. Let $s_i, s_j \in \{0,1\}$ be the signals at **i** and **j**, respectively. Then the following lemmas hold:

Lemma 6. *If cubes A and B satisfy relations $i = s_i$ and $j = s_j$ respectively, then the cube $C = A\phi B$ satisfies relation $(i = s_i) \wedge (j = s_j)$.*

Lemma 7. *Let S_i, S_j be sets of cubes satisfying $i = s_i$, $j = s_j$, respectively, then all cubes of the set $S_i \cup S_j$ satisfy relation $(i = s_i) \vee (j = s_j)$, while all cubes of the set $S_i \phi S_j$ satisfy relation $(i = s_i) \wedge (j = s_j)$.*

Our procedure for constructing a test set for all single stuck-at-faults consists of the following steps.

1. We construct covers for signals $S_1 \cup S_0$. To do this it is necessary to write $q + 1$ logical relations for the test signal transfer from input lines of the OR gate to the output line k (Fig. 9).

$$\begin{aligned}
&(1 = D) \wedge (2 = 0) \wedge \quad \ldots \quad \wedge (q = 0) \rightarrow (y = D),\\
&(1 = 0) \wedge (2 = D) \wedge \quad \ldots \quad \wedge (q = 0) \rightarrow (y = D),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (28)\\
&(1 = 0) \wedge (2 = 0) \wedge \quad \ldots \quad \wedge (q = D) \rightarrow (y = D),\\
&(1 = C) \wedge (2 = C) \wedge \quad \ldots \quad \wedge (q = C) \rightarrow (y = C).
\end{aligned}$$

2. For each relation (i), $i = 1, 2, \ldots, q + 1$, construct the covers $P_{i1}, P_{i2}, \ldots P_{iq}$ for the output lines of the gate $(\mathrm{AND})_i$ $(i = 1, 2, \ldots, q)$.

3. Define cubes

$$Q_i = P_{i1} \phi P_{i2} \phi \ldots \phi P_{iq} \quad (i = 1, 2, \ldots, q + 1)$$

   Since the combinational circuit of Fig. 9 is irredundant, it cannot happen that any of the cube sets Q_i $(i = 1, 2, \ldots, q + 1)$ is empty.

4. By developing the test set Q_i we obtain a set T_i of test vectors. A test set for all single-stuck-at-faults is given by

$$T = T_1 \cup T_2 \cup \ldots \cup T_q \cup T_{q+1}.$$

The statement made in 3. is obvious. Namely, if a test set Q_i were empty, a certain combination of signals from $S_1 \cup S_0$ could not appear at the input lines of the OR gate. On the basis of Theorem 1, the single stuck-at-fault (1 or 0) at the corresponding input line of the OR gate cannot be detected, as well as the stuck-at-fault (1 or 0) of the output line of the OR gate. This would mean that some single faults in the combinational circuit of Fig.9 are undetectable.

The sum of products of a Boolean function can be realized by a two-level combinational circuit in NAND-NAND implementation. Obviously, the proposed procedure can be applied to these cases, as the following example shows.

Example 8. Let us determine a test set for all single stuck-at-faults for the combinational circuit of Fig. 10.

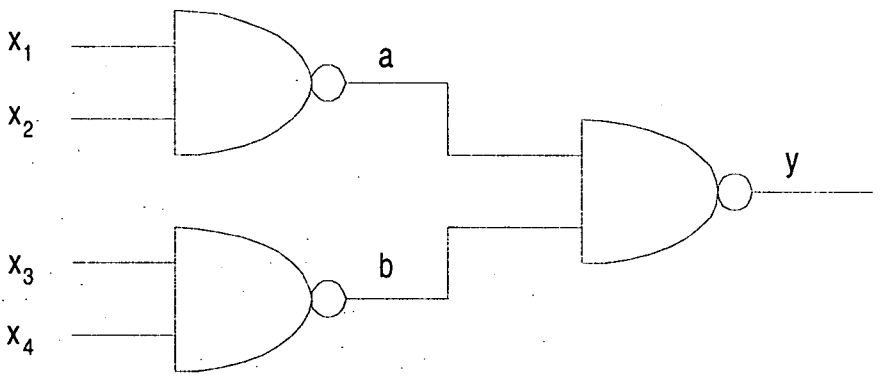


Figure 10: NAND-NAND implementation

Cascade NAND-NAND connection realizes an AND/OR function. Signals from the set $S_1 \cup S_0$ ($S_1 = \{C1, 1C\}$, $S_0 = \{DD\}$) are applied to the input lines of the NAND gate of the second level.

The following relations hold

$$(a = C) \wedge (b = 1) \rightarrow (y = D), \tag{29}$$

$$(a = 1) \wedge (b = C) \rightarrow (y = D), \tag{30}$$

$$(a = D) \wedge (b = D) \rightarrow (y = C). \tag{31}$$

The corresponding cube sets are

$$\begin{aligned} Q_1 &= \{11XX\}\phi\{XX0X, XXX0\} = \{110X, 11X0\}, \\ Q_2 &= \{0XXX, X0XX\}\phi\{XX11\} = \{OX11, X011\}, \\ Q_3 &= \{0XXX, X0XX\}\phi\{XX0X, XXX0\} = \\ &= \{0X0X, 0XX0, X00X, X0X0\}. \end{aligned}$$

Cubes determined by $Q_1 \cup Q_2 \cup Q_3$ are

$$110X$$

$$11X0$$

$$0X11$$

$$X011$$

$$0X0X$$

$$0XX0$$

$$X00X$$

$$X0X0.$$

We adopt $X = 1$, which gives the following vector sequence

$$1101$$

$$1110$$

$$0111$$

$$1011$$

$$0101$$

$$0110$$

$$1001$$

$$1010.$$

It is interesting that this vector sequence can be reordered in such way that each vector switches from 0-1 and from 1-0 during the test sequence. The obtained test set is minimal.

It should be pointed out here that in some combinational circuits, the detecting test set for single stuck-at-faults, detects at the same time all multiple stuck-at-faults. For example, this happens in fan-out free combinational circuits.

Combinational circuit of Fig. 9, which is considered in this paper, is a two-level irredundant combinational circuit. In the introduction we have already pointed out that any set of tests which detects all single stuck-at-faults also detects all multiple stuck-at-faults [7].

Of course, the length of the test set for all single stuck-at-faults in a combinational circuit, realized by the PAL architecture, depends on the Boolean function which is implemented.

We use the PAL architecture in constructing parity bit generators. The even parity bit P is a function of input variables $x_1, x_2, \ldots, x_n$ defined by

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \text{if the number of 1' is odd in } (x_1, x_2, \ldots, x_n) \\ 0, & \text{otherwise.} \end{cases}$$

This function can be represented in the form

$$P = f(x_1, x_2, \ldots, x_n) = \sum_{\alpha \in N} x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n} \tag{32}$$

where $\alpha = (\alpha_1 \alpha_2, \ldots, \alpha_n) \in \{0,1\}^n$, $N = \{\alpha \mid f(\alpha) = 1\}$ and $x^0 = \bar{x}$, $x^1 = x$.

The odd parity bit is defined as $\overline{f(x_1, x_2, \ldots, x_n)}$ and can be treated similarly.

As is known, function P has only one sum-of-product expression and it is just the quoted canonical sum-of-product expression. It has $2^n/2$ minterms. Length of test set for all single stuck-at-faults is 2^n, what means that we have exhaustive testing, (see, for example, [1]).

5 Comparing fault models

We shall discuss now the relevance of the proposed fault model by comparing it with existing models. In fact, we shall show that, by the use of the proposed procedure, one can detect also the faults appearing in most frequently used PLA fault models: stuck-at-faults, crosspoint faults and bridging faults.

First, we shall show that a number of classical stuck-at-faults can be detected using the proposed procedure for the detection a multiple fault appearing in programmable points of PLA.

Within the classical stuck-at-fault model we can treat the PLA as a two-level circuit implementation and the typical stuck-at faults appear at input, input inverters, product lines and output lines. We shall show that in our model these faults are detected as well.

In order to simplify the analysis but without the loss of generality, let us consider in our model the detectable fault α_i $(i = 1, 2, \ldots, m)$ which consists of stuck-at-faults located at r programmable points of the horizontal line $i(1)$, $i = 1, 2, \ldots, m$ (we assume that the multiple fault ai consists of single stuck-at faults at the first r programmable point). In these points test signal of values C and D (depending of the fault type) are generated. Obviously, test cubes detecting fault α_i on the line $i(1)$, have rank r, i.e. they have r coordinates of value 0 or 1, while in the last $n-r$ coordinates we have variable X $(X \in \{0,1\})$.

Obviously, the cube which detects the fault α_i will detect also the stuck-at-fault $i(1)/s$, $s \in \{0,1\}$, $i = 1, 2, \ldots, m$, since we get $i(1) = C$ and $i(1) = D$. The fault $i(1)/s$ is a dominant fault w.r.t. classical stuck-at-faults located at first r inputs of PLA. The cube which detects the fault α_i will detect also the stuck-at-faults at r inputs of PLA.

Since the fault α_i is detectable, the test signals of value C or D are propagated from line $i(1)$, $i = 1, 2, \ldots, m$, to the output line $h(2)$, $h = 1, 2, \ldots, p$ during testing. Suppose that the horizontal line $i(1)$, $i = 1, 2, \ldots, m$ in the AND array crosses the output line of PLA $h(2)$, $h = 1, 2, \ldots, p$ in the OR array, which means that this lines have a crossing point with coordinates (h, i). Obviously, test vectors detecting the fault α_i on the output line $i(1)$ in the AND array, detect this fault at each output line $h(2)$, $h = 1, 2, \ldots, p$, as well.

This analysis can be carried out for other distributions of a multiple fault in PLA, located in programmable points. Of course, the set of stuck-at-faults, appearing in the classical stuck-at-fault model and detected by the use of our model for a multiple fault, depends on the distribution (configuration) of that fault, in particular, on whether the fault is located in the AND or in the OR array or in both. In this analysis we can use properties of dominant and equivalent faults.

By a similar analysis we can conclude that the test vectors, detecting a multiple fault in PLA following the proposed procedure, detect also a number of bridging faults.

In the sequel we shall show that also a number of cross points faults are detected by the proposed procedure for detection of a multiple fault.

Faults of types G, D, S and A for the AND/OR implementation of PLA have been defined in the introduction. The AND gates of the AND array will be denoted by $(\text{AND})_i$, $i = 1, 2, \ldots, m$, while the OR gates of the OR array will be denoted by $(\text{OR})_i$, $i = 1, 2, \ldots, p$.

The fault of type G (growth fault) appears in the case when the line j ($j = 1, 2, \ldots, n$) for the gate $(\text{AND})_i$, $i = 1, 2, \ldots, m$, is broken. The brake of this input line causes that the implicant realizing the gate $(\text{AND})_i$ does not contain the input variable x_j corresponding to the broken line j.

As already described, when applying the procedure for detecting the fault α_i located at r programmable points of the horizontal line $i(1)$, $i = 1, 2, \ldots, m$, at output line $i(1)$ of the gate $(\text{AND})_i$, $i = 1, 2, \ldots, m$, we get test signals of the value C or D. Signals $i(1) = C$ and $i(1) = D$ are dominant w.r.t. the signals at r input lines $1, 2, \ldots, r$ of the gate $(\text{AND})_i$. Test vectors detecting the fault α_i detect also r faults of type G $1/1, 2/1, \ldots, r/1$.

The fault of type D (disappearance fault) appears in the case when an output line of the gate $(\text{AND})_i$ is broken. If this line represents the input of the gate $(\text{OR})_i$, $i = 1, 2, \ldots, p$, then the function f_i, realizing this gate, does not contain the implicant corresponding to the broken line. The fault of the broken line $i(1)$ will be denoted by $i(1)/1$.

When detecting the fault α_i test signals of values C and D are propagated from the line $i(1)$ $i = 1, 2, \ldots, m$ to the corresponding output line $i(2)$, $i = 1, 2, \ldots, p$ of PLA. Single fault of type D will be detected in the case when the proposed procedure generates the test signal value C at the line $i(1)$.

A similar analysis can be carried out for the faults of types S and A.

6 Conclusion

In the proposed fault detection procedure for combinational circuits, test signals have values from the set $\{0, 1, C, D\}$, while no line in the combinational circuit can be in state X. Obviously, this fact simplifies the testing procedure. In applying the proposed procedure we go only forward, thus avoiding the problem of unsuccessful steps, which appear in structural testing methods. Writing logical relations for the test signal propagation is easier, since one uses only operation "$\wedge$". Unlikely to

the method for combinational circuit testing mentioned in the introductory part, the variable X can have only two values ($X \in \{0,1\}$). If, for a given single or multiple fault, only a part of the test vector set is determined, the computation can be considerably reduced.

In the proposed fault detecting procedure test signals also get values from the set $\{0,1,C,D\}$. Test signals are assigned to faulty programmable elements, which form a multiple fault. Logical relations describing the test signal propagation from the source points to the output lines of PLA are constructed. While executing the testing procedure no programmable point of PLA can have an undefined signal value, i.e. to be in state X, which makes the procedure smoother to a great extent. One should point out that the proposed procedure is related to multiple fault detection, which leaves the space for its development in direction of getting a test set which detects all multiple faults which can appear in PLA.

We shall mention some standard fault models which are used to describe faulty PLA. The typical models used in PLA testing are: stuck-at-faults, crosspoint faults and bridging faults. The fault model, proposed in this paper, differs from the above mentioned models. It enables a PLA testing procedure having some advantages in relation to standard testing methods.

The stuck-at-faults model is a classical model for describing a faulty PLA. We have noted in the introduction that the following faults appear at lines of PLA: input inverters stuck-at-1 and 0, AND gate inputs and outputs stuck-at-1 and 0, OR gate inputs and outputs stuck-at-1 and 0 and output inverters stuck-at-1 and 0. It is well-known that while using the stuck-at-fault model a number of crosspoint faults as well as a number of bridging faults is detected. The starting point of the proposed procedure for detecting a multiple fault is the fact that the fault appears in programmable points of PLA, while the produced PLA corresponds to the designed one with the hypothesis that no mistakes in designing PLA can occur. Such a model reflects to a greater extent a real PLA. One should also point out that by the proposed procedure for detecting a multiple fault also a number of stuck-at-faults and bridging faults are detected as well. The number of detected stuck-at-faults, mentioned above, depends on the arrangement of the multiple fault in PLA.

The proposed procedure differs from the one used in the crosspoint faults model in which the faults of types G, D, S and A appear, where we test for incorrect logical connection in the AND and OR arrays. Obviously, the model involving the faults of types G, D, S and A is rather complicated one, while in our model we have only the faults of the type of disconnection of a programmable element or the faults of the type of a short connection of a programmable element. This means that crosspoints, in which faulty elements are located, obtain logical values 1 and 0.

With proposed procedure for multiple fault detection in the PLA, all test vectors are determined. In determining test vectors for a single or multiple fault, the quantity of work depends on the structure of the fault, and its disposition in PLA.

In this paper the stress is laid on the detection of a multiple stuck-at-fault for PLA circuits. The proposed procedure is also applied to some specific circuits, such as the PAL architecture realizing a irreduntant Boolean function. In this paper we

have shown that the test set which detects all single stuck-at-faults detects at the same time all multiple stuck-at-faults.

The proposed procedure for detecting multiple fault in PLA is very simple since the signal propagation is described by simple logical relations. Let us also point out that no extra hardware is necessary. Proposed procedure for fault detection can be applied to fault detection in integrated circuits implementing Boolean functions which may appear in realization of a computer architecture.

References

[1] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1995.

[2] J.M. Acken, S.D. Millman, *Fault model evolution for diagnosis: accuracy vs. precision*, Proc. IEEE Custom Integrated Circuits Conference, 1992, pp. 13.4.1–13.4.4.

[3] V. Agarwal, *Easily Testable PLA Design, in Advances in CAD for VLSI*, vol. 5, VLSI Testing, T. Wiliams (ed.), Elsevier Science Publishers, Amsterdam, 1986.

[4] V.Agraval, C.Kime, K.Saluja, *A Tutorial on Built-in-Self-Test*, IEEE Design & Test of Computers, March 1993, pp. 73-80, June 1993, pp. 69–77, IEEE CS Press.

[5] D.B. Armstrong, *On Finding a Nearly Minimal Set on Fault Detection Tests for Combinational Logic Nets*, IEEE Trans. Electronic Computers EC-15, 1966, pp. 66–73.

[6] P. Bibilo, N. Kirienko, *Block Synthesis of Combinational Circuits in the Basis of PLA and Library Gates*, The International Workshop on Discrete-Event System Design, June 27-29, 2001, Poland.

[7] M.A.Breuer, A.D. Friedman, *Diagnosis & Reliable Design of Digital Systems*, Computer Science Press, Inc., p. 68, 1976.

[8] C.W. Cha, W.E. Donath, F. Ozguner, *9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits*, IEEE Trans. on Computers, Vol. C-27, No. 3, March 1978, pp. 193–200.

[9] H.Fujiwara, S.Tolda, *The Complexity of Faults Detection Problems for Combinational Logic Circuits*, IEEE Trans. Comput., pp. 550–560 , June 1982.

[10] P. Goel, *An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits*, IEEE Trans. on Computers, Vol. C-30, No. 3, March 1981, pp. 215–221.

[11] I. Kohavi, Z.Kohavi, *Detection of Multiple Faults in Combinational Logic Networks*, IEEE Transactions on Computers, pp. 556–568, June 1972.

[12] K. Lai, P.K. Lala, *Multiple Fault Detection in Fan-Out Free Circuits Using Minimal Single Fault Test Set*, IEEE Trans. Comput. June 1996, pp. 763–765.

[13] P.K. Lala, *Digital Circuits Testing and Testability*, Academic Press, London 1997.

[14] S. Mourad, Y. Zorian, *Principles of Testing Electronic Systems*, John Wiley & Sons, 2000.

[15] K.S. Ramanatha, N.N. Biswas, *Design of Crosspoint-Irredundant PLAs Using Minimal Number of Control Inputs*, IEEE Trans. of Comp., Vol. 37, No. 9, September 1988, pp. 1130–1134.

[16] R.K. Ranjan, *Dynamic Reordering in a Bredth-First Manipulation Based BDD Package: Chalanges and solution* Proc. Of the 1997 IEEE Int. Conf. on Computer Design: VLSI in Computers & processors, ICCD'97, Austin, Texas, October 1977, pp. 17–26.

[17] J.P. Roth, *Diagnosis of Automata Failures: A Calculus and a Method*, IBM Journal, 1966, pp. 278–291.

[18] J.E. Smith, *Detection of Faults in Programmable Logic Arrays*, IEEE Trans. on Computers., vol.C-28, No 11. November 1979, pp. 845–853.

[19] C.E. Stroud, J.R. Bailey, J.R. Emmert, *A New Method for Testing Reprogrammable PLAs*, Journal of Electronic Testing, December 2000, pp.635–640.

Received April, 2003

Acta Cybernetica **16** (2004) 567–577.

A Hierarchy Theorem for Regular Languages over Free Bisemigroups

Zoltán L. Németh*

Abstract

In this article a question left open in [2] is answered. In particular, we show that it is essential that in the definition of parenthesizing automata an arbitrary number of parentheses can be used. Moreover, we prove that the classes Reg_m of languages accepted by a parenthesizing automaton with at most m pairs of parentheses form a strict hierarchy. In fact, this hierarchy is proper for all alphabets.

1 Introduction

A bisemigroup is set equipped with two associative operations. In [2], the notion of parenthesizing automaton operating on elements of free bisemigroups was introduced. The elements of free bisemigroups can be represented by labelled series-parallel biposets, or sp-biposets, for short. Biposets are sets with two partial order relations and a labelling function defined on them. A biposet is series-parallel if it can be obtained from the singleton biposets by the two associative operations corresponding to the order relations, called the series product ($\bullet$) and the parallel product ($\circ$). Hence, subsets of free bisemigroups accepted by parenthesizing automata will be called regular sp-biposet languages here. This concept of regularity proved to be appropriate for the characterization of algebraic recognizability, which is a very general notion well established in a universal algebraic setting: recognizability means to be recognized by a homomorphism into a finite bisemigroup.

An important feature of parenthesizing automata is that it is allowed to use any finite number of pairs of parentheses. The question emerges naturally if this feature is really necessary, or the number of parentheses can be bounded. In other words, we want to know whether there is a number K such that each regular sp-biposet language can be accepted by a parenthesizing automaton with at most K pairs of parentheses. This article gives the answer to this question. We show that no such K exists. Moreover, if Reg_m denotes the class of all regular sp-biposet languages that can be accepted by an automaton with $m \geq 0$ pairs of parentheses, then the classes $\mathsf{Reg}_0 \subsetneq \mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2 \subsetneq \ldots$ form a strict hierarchy. Furthermore,

*Dept. of Computer Science, University of Szeged, P.O.B. 652, 6701 Szeged, Hungary. E-mail: zlnemeth@inf.u-szeged.hu

we prove that this hierarchy is proper even when we consider languages over any fixed alphabet Σ.

This work can be seen principally as an addition to the paper of Ésik and the author on higher dimensional automata [2, 3]. (The latter is the journal version of the former.) Here we only briefly enumerate some related papers and refer to [3], where a whole section is devoted to a detailed comparison.

The theory of automata on biposets is closely related to the work of Lodaya and Weil [10, 11, 12] and Kuske [8, 9] on poset (partially ordered set) languages and on automata on (labelled posets). They studied the class of N-free (or in their terminology series-parallel) posets from the motivation of modelling concurrency. It is known that the N-free posets represent the elements of the free "semi-commutative bisemigroups", i.e., of the algebras equipped with an associative and an associative and commutative operation [4].

Automata and languages over free bisemigroups (more precisely, free bisemigroups with identity, called binoids or bimonoids) have also been studied in Hashiguchi et al., see [5, 6, 7]. But there the elements of the free binoid over Σ are represented by ordinary words in "standard form", which are the shortest expressions representing them over the extended alphabet $\Sigma \cup \{\bullet, \circ, (,)\}$. Ordinary finite automata are used to accept binoid languages. In [6] and [7], the notion of regular binoid expression is introduced defining the least class of binoid languages that contains the finite languages, and closed under the operations of union, the two product operations on languages, and the two iterations operations corresponding to the products. This class corresponds to the class of birational sp-biposet languages in [3]. The main result of [6] and [7] is that the binoid languages over Σ denoted by regular binoid expressions are those binoid languages, whose elements (in standard form) constitute regular word languages over the extended alphabet $\Sigma \cup \{\bullet, \circ, (,)\}$. This notion of regularity is less general than ours, e.g., the language of all biposets over an alphabet Σ cannot be represented by a regular binoid expression, but it can be accepted by a parenthesizing automaton with a single pair of parentheses. In fact, any language denoted by a regular binoid expression can be accepted with a single pair of parentheses.

2 Bisemigroups and biposets

In the sequel, n always denotes a positive integer and Σ a finite alphabet. We write $[n]$ for the set $\{1, 2, \ldots, n\}$. We use the notation Σ^n for the set of words over Σ of length n, and write Σ^* for the set of all words over Σ, as usual. Σ_m stands for any alphabet that has m letters. Let Ω denote a finite set of parentheses, its elements are usually written as $\langle_1, \rangle_1, \langle_2, \rangle_2, \ldots$. We assume that Ω is partitioned into the sets Ω_{op} of opening and Ω_{cl} of closing parentheses, which are in bijective correspondence.

We call a set equipped with n associative operations an *n-semigroup*. Automata operate on elements of some free algebra in general, on words in the classical case, i.e., on elements of free semigroups. Thus, if we want to generalize the notion of

automata to higher dimensions, it is natural to investigate how they operate on elements of free n-semigroups. In the following sections we only consider the case when $n = 2$, but all of our results can be extended to any nonnegative integer n in a straightforward way. When $n = 2$, we call an n-semigroup a *bisemigroup*. So a bisemigroup is a finite set S with two associative binary operations on it, called the *horizontal product* and the *vertical product*. We denote the two products by $\bullet$ and $\circ$, respectively.

We have several possible ways to describe the elements of a bisemigroup, freely generated by some finite alphabet Σ. First, they can be represented as terms over the extended alphabet $\Sigma \cup \{\bullet, \circ, (,)\}$ modulo associativity, or we can consider the (finite ordered) tree representations of these terms. But an element of a free bisemigroup can also be represented by a finite sp-biposet defined below.

Definition 2.1. *A* Σ-labelled biposet *or* biposet, *for short, is a structure* $(P, <_h^P, <_v^P, \lambda_P)$, *where* P *is a finite nonempty set of vertices,* $<_h^P$ *and* $<_v^P$ *are (irreflexive) partial orders on* P *and* $\lambda_P : P \to \Sigma$ *is a labelling function.*

We say that two biposets are *isomorphic* if there is a bijective function on the vertices that preserves the partial orders and the labelling. Below we will identify isomorphic biposets.

Suppose that $P = (P, <_h^P, <_v^P, \lambda_P)$ and $Q = (Q, <_h^Q, <_v^Q, \lambda_Q)$ are Σ-labelled biposets. Without loss of generality, assume that P and Q are disjoint. We define their *horizontal product* as $P \bullet Q = (P \cup Q, <_h^{P \bullet Q}, <_v^{P \bullet Q}, \lambda_{P \bullet Q})$, and their *vertical product* as $P \circ Q = (P \cup Q, <_h^{P \circ Q}, <_v^{P \circ Q}, \lambda_{P \circ Q})$, where

$$\begin{array}{lcllcl} <_h^{P \bullet Q} & = & <_h^P \cup <_h^Q \cup (P \times Q), & <_h^{P \circ Q} & = & <_h^P \cup <_h^Q, \\ <_v^{P \bullet Q} & = & <_v^P \cup <_v^Q, & <_v^{P \circ Q} & = & <_v^P \cup <_v^Q \cup (P \times Q), \end{array}$$

and the labellings are $\lambda_{P \bullet Q} = \lambda_{P \circ Q} = \lambda_P \cup \lambda_Q$.

It is obvious that both product operations are associative. Each letter $\sigma \in \Sigma$ may be identified with the singleton biposet labelled σ. Let $\mathrm{SPB}(\Sigma)$ denote the collection of biposets that can be generated from the singletons corresponding to the letters in Σ by the two product operations. The biposets in $\mathrm{SPB}(\Sigma)$ are called *series-parallel biposets*, or *sp-biposets*, for short.

As we mentioned above, the sp-biposets $\mathrm{SPB}(\Sigma)$ may serve as a possible description of the free bisemigroup generated by Σ. This fact is formulated in the following theorem.

Proposition 2.2. [1] $\mathrm{SPB}(\Sigma)$ *is freely generated by* Σ *in the variety of bisemigroups.*

3 Parenthesizing Automata

In this section we define parenthesizing automata that process sp-biposets in a sequential manner. The class of sp-biposet languages accepted by parenthesizing automata will be called regular sp-biposet languages.

Definition 3.1. *A (nondeterministic)* parenthesizing automaton *is a 9-tuple* $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$, *where* S *is the nonempty, finite set of* states, H *and* V *are the sets of* horizontal *and* vertical states, *which give a disjoint decomposition of* S, Σ *is the* input alphabet, Ω *is a finite set of parentheses, moreover,*

- $\delta \subseteq (H \times \Sigma \times H) \cup (V \times \Sigma \times V)$ *is the* labelled transition relation,
- $\gamma \subseteq (H \times \Omega \times V) \cup (V \times \Omega \times H)$ *is the* parenthesizing transition relation,
- $I, F \subseteq S$ *are the sets of* initial *and* final states, *respectively.*

We say that a biposet P has a *horizontal decomposition* into the horizontal product of biposets $P_1, P_2, \ldots, P_n$, $n \geq 2$, if $P = P_1 \bullet P_2 \bullet \ldots \bullet P_n$. A horizontal decomposition is said to be *maximal* if no component P_i, $(1 \leq i \leq n)$ has a horizontal decomposition. Vertical decompositions and maximal vertical decompositions are defined similarly.

The operation of the parenthesizing automata is based on the notion of the run, defined as follows.

Definition 3.2. *Suppose that* $P \in \mathrm{SPB}(\Sigma)$ *and* $p, q \in S$. *We say that* $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ has a run on P from p to q, *denoted* $[p, P, q]_{\mathcal{A}}$ *if one of the following conditions holds.*

(Base) $P = \sigma \in \Sigma$ *and* $(p, \sigma, q) \in \delta$.

(HH) $p, q \in H$ *and* P *has maximal horizontal decomposition* $P = P_1 \bullet \ldots \bullet P_n$, *where* $n \geq 2$, *and there exist* $r_1, \ldots, r_{n-1} \in S$, $r_0 = p$, $r_n = q$ *such that* $[r_{i-1}, P_i, r_i]_{\mathcal{A}}$, *for all* $i \in [n]$.

(VV) $p, q \in V$ *and* P *has maximal vertical decomposition* $P = P_1 \circ \ldots \circ P_n$, *where* $n \geq 2$, *and there exist* $r_1, \ldots, r_{n-1} \in S$, $r_0 = p$, $r_n = q$ *such that* $[r_{i-1}, P_i, r_i]_{\mathcal{A}}$ *for all* $i \in [n]$.

(HV) $p, q \in H$ *and* P *has maximal vertical decomposition* $P = P_1 \circ \ldots \circ P_n$, *where* $n \geq 2$, *and there exist* $\langle_k, \rangle_k \in \Omega$, $p', q' \in V$ *and* $(p, \langle_k, p'), (q', \rangle_k, q) \in \gamma$ *such that* $[p', P, q']_{\mathcal{A}}$ *holds.*

(VH) $p, q \in V$ *and* P *has maximal horizontal decomposition* $P = P_1 \bullet \ldots \bullet P_n$, *where* $n \geq 2$, *and there exist* $\langle_k, \rangle_k \in \Omega$, $p', q' \in H$ *and* $(p, \langle_k, p'), (q', \rangle_k, q) \in \gamma$ *such that* $[p', P, q']_{\mathcal{A}}$ *holds.*

An *accepting run* is a run from an initial state to a final state. The sp-biposet *language* $L(\mathcal{A})$ *accepted by the automaton* $\mathcal{A}$ is defined as the set of all labels of the accepting runs. Formally,

$$L(\mathcal{A}) \;=\; \{P \in \mathrm{SPB}(\Sigma) \mid \exists i \in I,\ f \in F \;:\; [i, P, f]_{\mathcal{A}}\}.$$

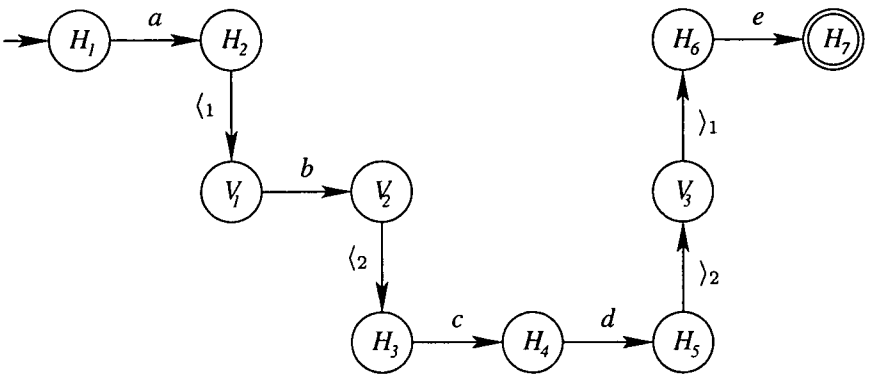


Figure 1: A parenthesizing automaton accepting $\{a \bullet (b \circ (c \bullet d)) \bullet e\}$.

Example 3.3. The automaton given on Figure 1 has a single initial state H_1 and a single final state H_7. The horizontal states are those labelled H_i and the vertical states those labelled V_j, for some i and j. This automaton accepts the singleton sp-biposet language consisting of the biposet $a \bullet (b \circ (c \bullet d)) \bullet e$. Indeed, there are runs $[H_1, a, H_2]_{\mathcal{A}}$, $[H_2, b \circ (c \bullet d), H_6]_{\mathcal{A}}$ and $[H_6, e, H_7]_{\mathcal{A}}$ corresponding to the horizontal decomposition of the biposet. Moreover, according to the case (HV) of Definition 3.2, the second run, $[H_2, b \circ (c \bullet d), H_6]_{\mathcal{A}}$ starts with a parenthesizing transition $(H_2, \langle_1, V_1)$ followed by a subrun $[V_1, b \circ (c \bullet d), V_3]_{\mathcal{A}}$ and ends with the transition $(V_3, \rangle_1, H_6)$. The existence of the subrun $[V_1, b \circ (c \bullet d), V_3]_{\mathcal{A}}$ comes from the existence of the runs $[V_1, b, V_2]_{\mathcal{A}}$ and $[V_2, c \bullet d, V_3]_{\mathcal{A}}$ that can be seen similarly. Several other examples of parenthesizing automata can be found in [3].

Recall from [3] that an sp-biposet language $L \subseteq \mathrm{SPB}(\Sigma)$ is called *recognizable* if there is a finite bisemigroup B, a homomorphism $h : \mathrm{SPB}(\Sigma) \to B$, and a set $F \subseteq B$ with $L = h^{-1}(F)$. Let $\mathsf{Rec}(\Sigma)$ and $\mathsf{Reg}(\Sigma)$ denote the set of recognizable and regular sp-biposet languages of $\mathrm{SPB}(\Sigma)$, respectively. Moreover, write Rec and Reg for the classes of all regular and recognizable sp-biposet languages, respectively. One of the main results of [3] shows that these two classes coincide.

Theorem 3.4. [3] $\mathsf{Rec} = \mathsf{Reg}$, *i.e., an sp-biposet language* $L \subseteq \mathrm{SPB}(\Sigma)$ *is recognizable iff* L *is regular.*

4 Hierarchy Theorems

In this session we present the main hierarchy results of the paper, formulated in Theorem 4.9 and 4.10. Let Reg_m denote the regular sp-biposet languages that can be accepted by a parenthesizing automaton with at most $m \geq 0$ pairs of parentheses. We will show that $\mathsf{Reg}_m \subsetneq \mathsf{Reg}_{m+1}$ for all $m \geq 0$. Moreover, if $\mathsf{Reg}_m(\Sigma)$ is the set of all regular sp-biposet languages over any fixed alphabet Σ that can be accepted by

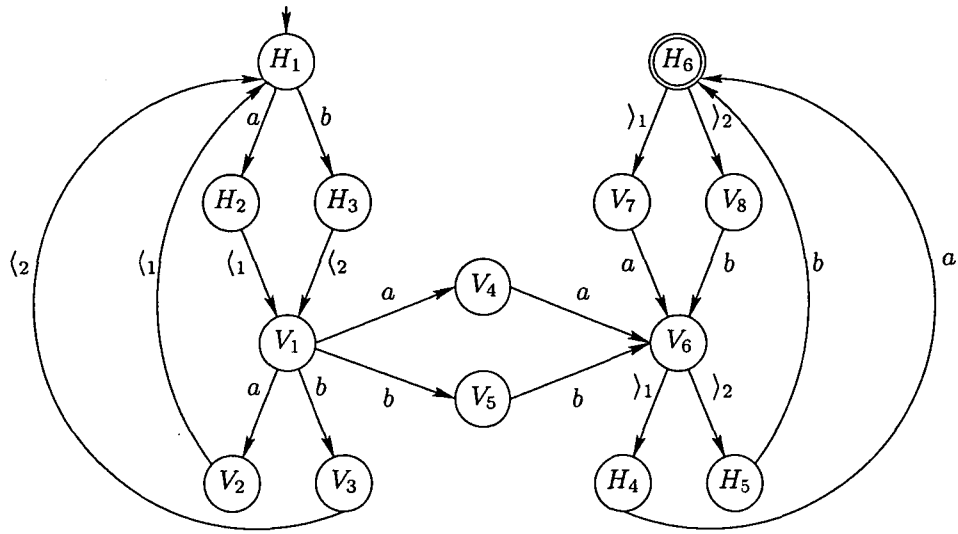


Figure 2: An automaton $\mathcal{A}_2$ accepting $\hat{L}$

an automaton with at most m pairs of parentheses, then $\mathsf{Reg}_m(\Sigma) \subsetneq \mathsf{Reg}_{m+1}(\Sigma)$, for all $m \geq 0$. Our first aim is to define a language $\hat{L}$ in $\mathsf{Reg}_2 \setminus \mathsf{Reg}_1$.

Definition 4.1. *For any words $u = \sigma_1\sigma_2 \ldots \sigma_n$ and $v = \rho_1\rho_2 \ldots \rho_n$ of even length over any alphabet Σ, let $P_{uv^{-1}}$ denote the biposet*

$$P_{uv^{-1}} = \sigma_1 \bullet (\sigma_2 \circ (\sigma_3 \bullet (\sigma_4 \circ \ldots (\sigma_{n-1} \bullet (\sigma_n \circ \rho_n) \bullet \rho_{n-1}) \ldots \circ \rho_4) \bullet \rho_3) \circ \rho_2) \bullet \rho_1.$$

Definition 4.2. *Let Σ_2 denote the two-letter alphabet $\Sigma_2 = \{a, b\}$, and let*

$$\begin{aligned} \hat{L} &= \bigcup_{i=1}^{\infty} \hat{L}_{2i}, \text{ where} \\ \hat{L}_2 &= \{\, \sigma \bullet (\sigma' \circ \sigma') \bullet \sigma \mid \sigma, \sigma' \in \Sigma_2 \,\}, \\ \hat{L}_{2i+2} &= \{\, \sigma \bullet (\sigma' \circ P \circ \sigma') \bullet \sigma \mid \sigma, \sigma' \in \Sigma_2 \text{ and } P \in \hat{L}_{2i} \,\}, \text{ for all } i \geq 1. \end{aligned}$$

Remark 4.3. Note that $\hat{L} = \{\, P_{ww^{-1}} \mid w \in \Sigma_2^*,\ w \text{ has even length} \,\}$.

Proposition 4.4. *$\hat{L}$ can be accepted by a parenthesizing automaton that has two pairs of parentheses, i.e., $\hat{L} \in \mathsf{Reg}_2(\Sigma_2)$.*

Proof. It is not hard to see that the automaton $\mathcal{A}_2$ in Figure 2 accepts $\hat{L}$. □

Our next aim is to show that $\hat{L}$ is not in $\mathsf{Reg}_1(\Sigma_2)$. In order to see this we must describe the runs of an automaton step by step. So we introduce the notion of generalized state of an automaton as follows.

Definition 4.5. *A* generalized state *of a parenthesizing automaton* $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ *is a pair* (q, ω), *where* $q \in S$ *and* $\omega \in \Omega_{op}^*$.

Definition 4.6. *We say that an automaton* $\mathcal{A}$ has a transition *from a generalized state* (q_1, ω_1) *to a generalized state* (q_2, ω_2) *with respect to the symbol* $\alpha \in \Sigma \cup \Omega$ *if one of the following conditions holds:*

(i) $\alpha \in \Sigma$, $\omega_1 = \omega_2$ *and* $(q_1, \alpha, q_2) \in \delta$, *or*

(ii) $\alpha \in \Omega_{op}$, $\omega_2 = \omega_1 \alpha$ *and* $(q_1, \alpha, q_2) \in \gamma$, *or*

(iii) $\alpha \in \Omega_{cl}$, $\omega_1 = \omega_2 \bar{\alpha}$, *where* $\bar{\alpha}$ *is the opening pair of the closing parenthesis* α *and* $(q_1, \alpha, q_2) \in \gamma$.

Remark 4.7. Thus, every run $[p, P, q]_{\mathcal{A}}$ corresponds to a sequence of transitions between generalized states

$$(q_0, \varepsilon) \overset{\alpha_1}{\vdash} (q_1, \omega_1) \overset{\alpha_2}{\vdash} (q_2, \omega_2) \overset{\alpha_3}{\vdash} \dots \overset{\alpha_n}{\vdash} (q_n, \varepsilon),$$

where $q_0 = p, q_n = q$ and $(q_i, \alpha_i, q_{i+1}) \in \delta \cup \gamma$ for $i \in [n]$, and ω_i is the sequence of opened but not yet closed parentheses after the first i steps of the run.

Proposition 4.8. *There is no parenthesizing automaton with a single pair of parentheses accepting* $\hat{L}$, *i.e.,* $\hat{L} \notin \mathsf{Reg}_1(\Sigma_2)$.

Proof. On the contrary, suppose that there is a parenthesizing automaton $\mathcal{A} = (S, H, V, \Sigma_2, \Omega_1, \gamma, \delta, I, F)$ accepting $\hat{L}$ with $\Omega_1 = \{\langle, \rangle\}$. Let n be an even integer greater than $|V|$. The number of all biposets $P_{ww^{-1}} \in \hat{L}$, where w is of length n, is 2^n. Thus, since $2n - 2 \leq 2^n$ holds for all n, either there are n biposets accepted between horizontal states, or there are n biposets accepted between vertical states. For simplicity, we assume the former case. This is not a real restriction, since in the other case our proof would be essentially the same, only an opening and a closing parenthesizing transition needs to be added before and after the runs.

Thus, let us take n distinct words $w_j \in \Sigma_2^n$, $j \in [n]$, and consider the biposets

$$P_{w_j w_j^{-1}} = \sigma_1^j \bullet (\sigma_2^j \circ (\sigma_3^j \bullet (\sigma_4^j \circ \dots (\sigma_{n-1}^j \bullet (\sigma_n^j \circ \sigma_n^j) \bullet \sigma_{n-1}^j) \dots \circ \sigma_4^j) \bullet \sigma_3^j) \circ \sigma_2^j) \bullet \sigma_1^j,$$

where $\sigma_1^j \sigma_2^j \dots \sigma_n^j = w_j$. Now, each $P_{w_j w_j^{-1}}$ is in $\hat{L}$, by definition, but, as we shall see, $\mathcal{A}$ must accept biposets that do not belong to $\hat{L}$.

Indeed, each accepting run of $\mathcal{A}$ on any $P_{w_j w_j^{-1}}$ must have the form

$$(q_0,\varepsilon) \overset{\sigma_1^j}{\vdash} (q_1,\varepsilon) \overset{\langle}{\vdash} (q_2,\langle) \overset{\sigma_2^j}{\vdash} (q_3,\langle) \overset{\langle}{\vdash} (q_4,\langle\langle) \overset{\sigma_3^j}{\vdash} \dots \overset{\sigma_{n-1}^j}{\vdash} (q_{2n-3},\langle^{n-2}) \overset{\langle}{\vdash}$$
$$(q_{2n-2},\langle^{n-1}) \overset{\sigma_n^j}{\vdash} (q_{2n-1},\langle^{n-1}) \overset{\sigma_n^j}{\vdash} (q_{2n},\langle^{n-1}) \overset{\rangle}{\vdash} (q_{2n+1},\langle^{n-2}) \overset{\sigma_{n-1}^j}{\vdash}$$
$$(q_{2n+2},\langle^{n-2}) \overset{\rangle}{\vdash} \dots \overset{\sigma_2^j}{\vdash} (q_{4n-4},\langle) \overset{\rangle}{\vdash} (q_{4n-3},\varepsilon) \overset{\sigma_1^j}{\vdash} (q_{4n-2},\varepsilon),$$

where $q_0 \in I \cap H$ and $q_{4n-2} \in F \cap H$.

For our investigation the main point is that after processing the "first half" of $P_{w_j w_j^{-1}}$, (i.e., after $2n-1$ transitions) the automaton enters a generalized state $(q_{2n-1},\langle^{n-1})$, where q_{2n-1} is vertical and after an additional $2n-1$ transitions, the run ends in (q_{4n-2},ε).

Given $P_{w_j w_j^{-1}}$, let i_j, v_j and f_j denote the states q_0, q_{2n-1}, and q_{4n-2}, respectively, which appear in the above accepting run of $\mathcal{A}$ on the biposet $P_{w_j w_j^{-1}}$. Moreover, let us abbreviate the transition sequences determined by the first $2n-1$ and the second $2n-1$ transitions by

$$(i_j,\varepsilon) \overset{P_{w_j *}}{\vdash} (v_j,\langle^{n-1}) \text{ and } (v_j,\langle^{n-1}) \overset{P_{* w_j^{-1}}}{\vdash} (f_j,\varepsilon),$$

respectively.

Now we have n vertical states $v_1, v_2, \dots, v_n$, but we have chosen $n > |V|$, so there must be two indices $k \neq l$ such that $v_k = v_l$. Hence, the transition sequence

$$(i_k,\varepsilon) \overset{P_{w_k *}}{\vdash} (v_k,\langle^{n-1}) = (v_l,\langle^{n-1}) \overset{P_{* w_l^{-1}}}{\vdash} (f_l,\varepsilon)$$

corresponds to a valid run of $\mathcal{A}$, showing that $P_{w_k w_l^{-1}}$ is accepted by $\mathcal{A}$. But $P_{w_k w_l^{-1}} \notin \hat{L}$, a contradiction. Thus, no $\mathcal{A}$ with a single pair of parentheses can accept $\hat{L}$, so $\hat{L} \notin \mathsf{Reg}_1(\Sigma_2)$. □

The previous theorem can be extended to any $m \geq 1$ as follows.

Theorem 4.9. *For all $m \geq 1$ there exists a language $\hat{L}(\Sigma_m)$ that can be accepted by an automaton with m but not with $m-1$ pairs of parentheses. Thus, the classes $\mathsf{Reg}_0 \subsetneq \mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2 \subsetneq \dots$ form a strict hierarchy of regular (i.e., recognizable) sp-biposet languages.*

Proof. Our claim is trivial for $m = 1$, and $\mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2$ was shown in Proposition 4.4 and Proposition 4.8. As for $m \geq 3$, we show how the proofs of these two propositions can be generalized. Let

$$\hat{L}(\Sigma_m) = \{\, P_{ww^{-1}} \mid w \in \Sigma_m^*,\ w \text{ has even length}\}.$$

Note that $\hat{L} = \hat{L}(\Sigma_2)$. In Proposition 4.4, the automaton $\mathcal{A}_2$ accepting $\hat{L}$ can easily be generalized to an automaton $\mathcal{A}_m$ accepting $\hat{L}(\Sigma_m)$, by using m different parentheses corresponding to the m letters.

In order to see that $\hat{L}(\Sigma_m) \notin \mathsf{Reg}_{m-1}$, suppose on the contrary that $\hat{L}(\Sigma_m) = L(\mathcal{A})$ for some automaton $\mathcal{A} = (S, H, V, \Sigma_m, \Omega_{m-1}, \gamma, \delta, I, F)$, where $\Omega_{m-1} = \{\langle_1, \rangle_1, \langle_2, \rangle_2, \ldots, \langle_{m-1}, \rangle_{m-1}\}$. We choose pairwise different words $w_1, w_2, \ldots, w_n$ in Σ_m^n as before, but this time we give the value of n later.

We notice that after reading the first half of $P_{w_j w_j^{-1}}$, where $P_{w_j w_j^{-1}}$ is defined as in the proof of Theorem 4.8, automaton $\mathcal{A}$ is necessarily in a generalized state

$$(v_j, \langle_{i_1} \langle_{i_2} \cdots \langle_{i_{n-1}}), \qquad (*)$$

where $v_j \in V$ and $\langle_{i_k} \in \Omega_{m-1}$ for all $k = 1, 2, \ldots, n-1$.

But the number of this type of generalized states, $|V| \cdot (m-1)^{n-1}$ is asymptotically less than m^n, the number of words w_j of length n, which is the number of all biposets of the form $P_{w_j w_j^{-1}}$. Thus, in the same way as above, n can always be chosen such that $\mathcal{A}$ accepts at least one biposet $P_{w_j w_k^{-1}}$ for some $w_j \neq w_k$. □

In the proof of Theorem 4.9 we used the m-letter alphabet Σ_m to show that $\mathsf{Reg}_{m-1}(\Sigma_m) \subsetneq \mathsf{Reg}_m(\Sigma_m)$. However this proper inclusion holds for every Σ.

Theorem 4.10. *For all alphabets Σ the classes $\mathsf{Reg}_0(\Sigma) \subsetneq \mathsf{Reg}_1(\Sigma) \subsetneq \mathsf{Reg}_2(\Sigma) \ldots$ form a strict hierarchy in $\mathsf{Reg}(\Sigma)$.*

Proof. It is sufficient to prove this claim for a one-letter alphabet. So assume that $\Sigma = \Sigma_1 = \{a\}$ and suppose that $m \geq 1$. Let h denote the bisemigroup homomorphism $\mathrm{SPB}(\Sigma_m) \to \mathrm{SPB}(\Sigma_1)$ determined by the assignment

$$a_i \mapsto \underbrace{a \bullet a \bullet \ldots \bullet a}_{i \text{ times}}, \text{for all } a_i \in \Sigma_m.$$

We claim that the language $h(\hat{L}(\Sigma_m))$ is in $\mathsf{Reg}_m(\Sigma_1) \setminus \mathsf{Reg}_{m-1}(\Sigma_1)$.

Indeed, it is not hard to modify the automaton $\mathcal{A}_m$ in the proof of Theorem 4.9 to accept $h(\hat{L}(\Sigma_m))$ instead of $\hat{L}(\Sigma_m)$. On the other hand, after reading the "first half" of a biposet $h(P_{w_j w_j^{-1}})$, any parenthesizing automaton with $m-1$ pairs of parentheses must be in a generalized state (*) as before. Thus, the same cardinality argument can be applied to show that $h(\hat{L}(\Sigma_m))$ is not in $\mathsf{Reg}_{m-1}(\Sigma_1)$. □

5 Conclusions and further work

In this paper we dealt with the descriptional complexity of regular sp-biposet languages measured in terms of the least number of parentheses that an automaton needs to accept them. We have shown that with more pairs of parentheses strictly larger classes of regular sp-biposet languages can be accepted.

Recall from [3] that BRat denotes the class of birational languages which is the least class of sp-biposet languages containing the finite languages and closed under

union, horizontal and vertical product and horizontal and vertical iteration. We can prove that $\mathsf{BRat} \subseteq \mathsf{Reg}_1$. The proof relies on the fact that every birational language has bounded alternation depth, i.e., for each birational sp-biposet language L there is a bound $K \geq 0$ such that every element in L has at most K pairs of nested parentheses. BD is the class of bounded alternation depth languages. Thus, if a parenthesizing automaton $\mathcal{A}$ accepts a birational language, then the successful runs of $\mathcal{A}$ must have a bounded number of opened but not yet closed parentheses at any time. So one can construct an equivalent automaton $\mathcal{A}'$ with a single pair of parentheses by storing the information about all such opened parentheses in the (inner) states. This fact together with the equation $\mathsf{BRat} = \mathsf{Reg} \cap \mathsf{BD}$ (see [3]) leads to the characterization of BRat as $\mathsf{BRat} = \mathsf{Reg}_1 \cap \mathsf{BD}$.

An open problem that seems to be difficult to solve is the decidability status of the question whether a given regular language appears in a certain level of the hierarchy. It would also be interesting to find algebraic or logical characterizations of the levels of our hierarchy.

Acknowledgement

The author is indebted to Zoltán Ésik for many valuable comments and suggestions.

References

[1] Z. Ésik. Free algebras for generalized automata and language theory. *RIMS Kokyuroku 1166*, Kyoto University, Kyoto, 2000, 52–58.

[2] Z. Ésik and Z. L. Németh. Automata on series-parallel biposets. In: proc. *DLT'01*, LNCS 2295, Springer, 2002, 217–227.

[3] Z. Ésik and Z. L. Németh. Higher dimensional automata, *J. of Automata, Languages and Combinatorics*, 9(2004), 3–29.

[4] J. L. Gischer. The equational theory of pomsets. *Theoret. Comput. Sci.*, 61(1988), 199–224.

[5] K. Hashiguchi, S. Ichihara and S. Jimbo. Formal languages over free binoids. *J. Autom. Lang. Comb.*, 5(2000), 219–234.

[6] K. Hashiguchi, Y. Wada and S. Jimbo. Regular binoid expressions and regular binoid languages. *Theoret. Comput. Sci.*, 304(2003), 291–313.

[7] K. Hashiguchi, Y. Sakakibara and S. Jimbo. Equivalence of regular binoid expressions and regular expressions denoting binoid languages over free binoids. *Theoret. Comput. Sci.*, 312(2004), 251–266.

[8] D. Kuske. Infinite series-parallel posets: logic and languages. In: proc. *ICALP 2000*, LNCS 1853, Springer, 2001, 648–662.

[9] D. Kuske. Towards a language theory for infinite N-free pomsets. *Theoret. Comput. Sci.*, 299(2003), 347–386.

[10] K. Lodaya and P. Weil. Kleene iteration for parallelism. In: proc. *FST & TCS'98*, LNCS 1530, Springer-Verlag, 1998, 355–366.

[11] K. Lodaya and P. Weil. Series-parallel languages and the bounded-width property. *Theoret. Comput. Sci.*, 237(2000), 347–380.

[12] K. Lodaya and P. Weil. Rationality in algebras with series operation. *Inform. and Comput.*, 171(2001), 269-293.

Received April, 2004

Acta Cybernetica **16** (2004) 579–585.

Hexagonal polyomino weak (1,2)-achievement games

Nándor Sieben*

Abstract

A version of polyomino achievement games is studied, in which the first player marks one cell and the second player marks two cells at each move. All polyominos but one on an infinite 2-dimensional hexagonal board are characterized to be weak winners or losers.

Keywords: polyomino, animal, achievement game

1 Introduction

Achievement games for polyominos have been introduced by Frank Harary [Gar, Ha1, Ha2, HH, Ha3]. They are generalizations of the well known game Tic-Tac-Toe, where the target shape can be some predetermined set of polyominos. The type of the board can vary as well. It can be a tiling of the plane by triangles [BH3] or hexagons [BH2]. The board can also have higher dimensional [HW, DS], it can be a Platonic solid [BH1] or the hyperbolic plane [Bod]. For further results see [HHS, HS1, HS2, HS3].

A *polyomino* or *animal* is a finite set of cells such that both the polyomino and its complement are connected through edges. We only consider polyominos up to congruence, that is, the location and direction of the polyomino on the board is not important.

In a *polyomino weak achievement game,* or *A-achievement game,* two players alternately mark previously unmarked cells of the board using their own colors. The first player (the maker) wins if he can mark a set of cells congruent to a given polyomino. The second player (the breaker) wins if she can prevent the maker from achieving the given polyomino. In a $(1,2)$-achievement game [Plu, Sie] the first player marks a single cell while the second player marks two cells at each move.

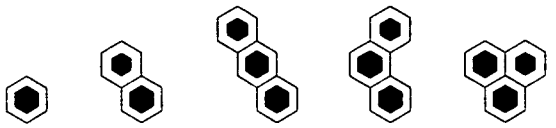
A polyomino is called a (*weak*) *winner* if the first player can always win the (weak) achievement game with the given polyomino. Otherwise the polyomino is called a *loser.*

In this paper we classify all animals but one as winners or losers in the weak $(1,2)$-achievement game on an infinite hexagonal board.

*Department of Mathematics and Statistics, Northern Arizona University, Flagstaff, AZ 86011-5717, USA, E-mail: nandor.sieben@nau.edu

2 Animals with at most three cells

There are five animals with fewer than four cells:

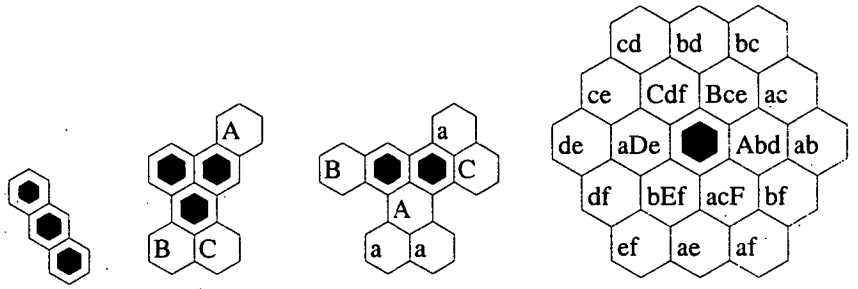


P_1 P_2 P_3 P_4 P_5

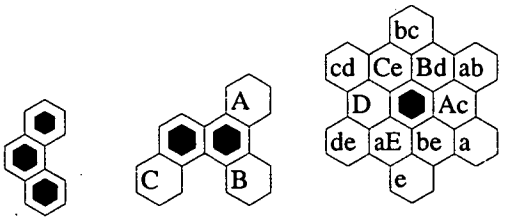
It is clear that P_1 and P_2 are winners.

Proposition 1. *The animals P_3 and P_4 are winners.*

Proof. The following are winning strategies for these animals:



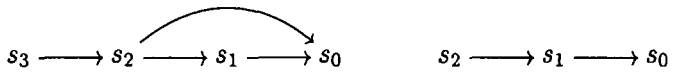
$P_3 = s_0$ s_1 s_2 s_3



$P_4 = s_0$ s_1 s_2

Cells with a solid block represent the marks of the maker. Cells with letters in them represent empty cells. The games start at situation s_3 and s_2 respectively and end at situation s_0. It is easy to see that no matter how the breaker marks two empty cells there is a letter which was not present in those two cells. Then the maker can mark the cell containing the capitalized version of this missing letter and achieve a situation with a smaller number. This situation is determined by the cells containing the missing letter.

The following figure shows the flowcharts of these games:

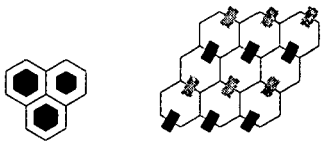


□

Proposition 2. *The animal P_5 is a loser.*

Proof. To show that an animal is a loser we can pave the board twice, using pairs of cells. We choose the two pavings such that every position of the given animal on the board contains a full pair of cells either from one of the pavings or from the other. The existence of such double paving allows the breaker to win by marking the two cells which are the pairs of the maker's mark in the two pavings. None of these cells can already be marked by the maker but it is possible that they are already marked by the breaker. In that case the breaker can mark any other cell.

The following figure shows P_5 and its double paving:



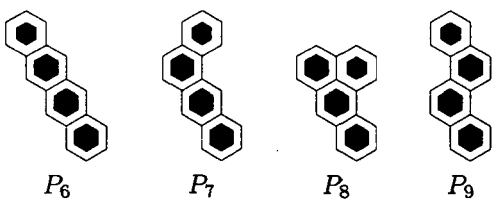
P_5

The reader can easily verify that the double paving has the required properties.

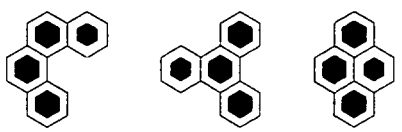
□

3 Animals with four cells

An animal containing a smaller losing animal is a loser itself. Hence the only possibility to find winners with four cells is to consider animals that are created from P_3 or P_4 by adding a single cell. There are eight such animals:



P_6 P_7 P_8 P_9



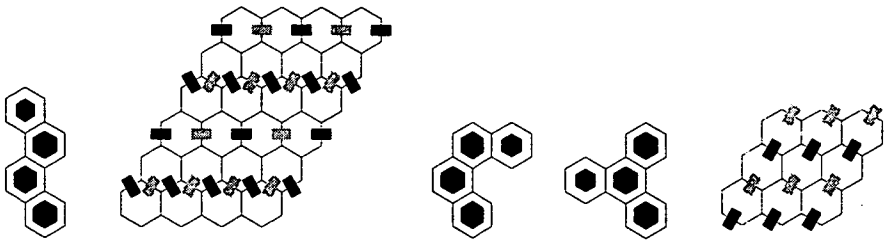
P_{10} P_{11} P_{12}

Proposition 3. *The animals P_8 and P_{12} are losers.*

Proof. All of these contain the loser P_5 as a subset and therefore they are losers themselves. □

Proposition 4. *The animals P_9, P_{10} and P_{11} are losers.*

Proof. These animals are losers because of the strategies based on the following double pavings:

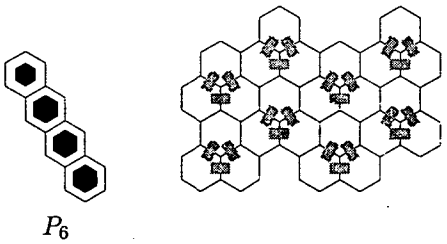


□

Proposition 5. *The animal P_6 is a loser.*

Proof. To show that an animal is a loser we can pave the board using triples of cells. We choose the paving such that every position of the given animal on the board contains at least two cells from one of the triples of the paving. The existence of such paving allows the breaker to win by marking the two cells which are in the same triple as the maker's mark.

The following figure shows the animal and its paving by triples:

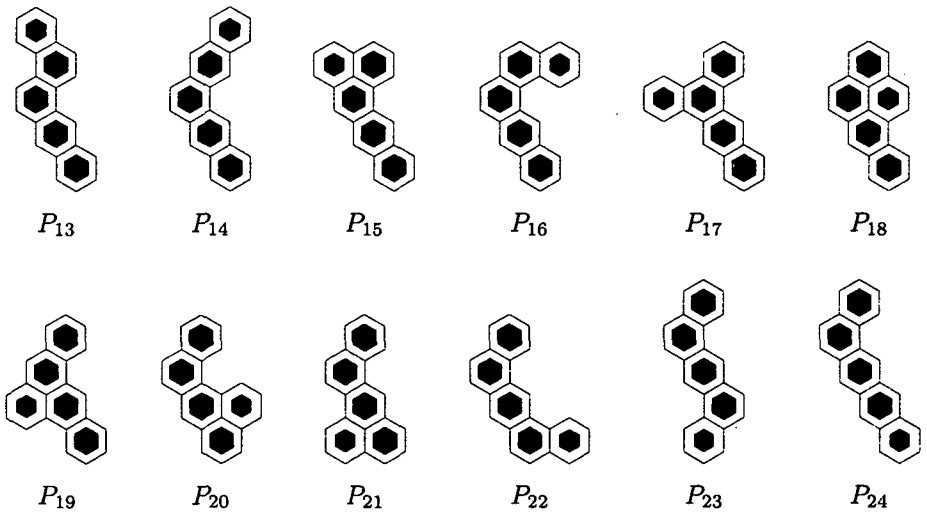


□

Animal P_7 remains a mystery. Playing the game suggests that it is a loser. Full analysis of the game tree shows that the maker wins if the breaker is required to mark cells that share an edge with an already marked cell. This shows that P_7 is a *paving winner*, that is, there is no winning strategy for the breaker based on a paving. We suspect that it is also a *pair partition winner*, that is, the breaker does not have a winning strategy based on a pairing even if disconnected pairs of cells are allowed.

4 Animals with five cells

The only possibility to find winners with five cells is to consider animals that are created from P_7 by adding a single cell. There are twelve such animals:

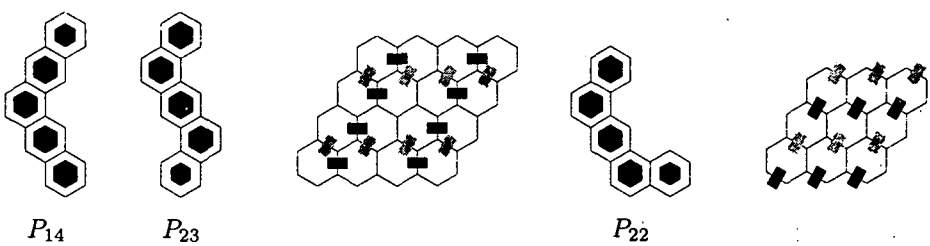


Proposition 6. *Animals P_{13}, P_{15}, P_{16}, P_{17}, P_{18}, P_{19}, P_{20}, P_{21} and P_{24} are losers.*

Proof. All of these animals contain one of the losers P_5, P_6, P_9, P_{10} or P_{11} as a subset and therefore they are losers themselves. □

Proposition 7. *The animals P_{14}, P_{22} and P_{23} are losers.*

Proof. All these animals are losers because of strategies based on the following double pavings:



□

5 Main result

Since all animals with five cells are losers, there cannot be any winners with more than four cells. So we have the following main result.

Theorem 8. *The only winning animals in the weak* $(1,2)$*-achievement game on the 2-dimensional infinite hexagonal board are the animals* P_1, P_2, P_3, P_4 *and possibly* P_7.

There are several questions to be answered. Is P_7 a winner or a loser? What are the winning pairs, triples or larger sets of animals? What are the handicap numbers of the losers, that is, how many extra marks does the maker need before the game starts to be able to win? What are the winners in the (1,2)-achievement game on triangular, and higher dimensional boards? What are the winners in the (2,3)-achievement game?

References

[Bod] J. Bode, *Strategien für Aufbauspiele mit mosaik-polyominos,* Doctoral dissertation, Technishen Universität Braunschweig.

[BH1] J. Bode, H. Harborth, *Achievement games on Platonic solids,* Bull. Inst. Combin. Appl. 23 (1998), 23–32.

[BH2] J. Bode, H. Harborth, *Hexagonal polyomino achievement,* Discrete Math. **212** (2000), np. 1–2, 5–18.

[BH3] J. Bode, H. Harborth, *Triangle polyomino set achievement,* Congr. Numer. **148** (2001), 97–101.

[DS] E. Deabay, N. Sieben, *Poliomino weak achievement games on 3-dimensional rectangular boards,* preprint.

[Gar] M. Gardner, *Mathematical games,* Sci. Amer. 240 (1979) 18–26.

[Ha1] F. Harary, *Achieving the Skinny animal,* Eureka **42** (1982) 8–14.

[Ha2] F. Harary, *Achievement and avoidance games for graphs,* Ann. Discrete Math. **13** (1982), 111–120.

[Ha3] F. Harary, *Is Snaky a winner?* Geombinatorics **2** (1993), 79–82.

[HH] F. Harary, H. Harborth, *Achievement and avoidance games with triangular animals,* Recreational Math. **18** (1985–86), 110–115.

[HHS] F. Harary, H. Harborth, M. Seeman, *Handicap achievement for polyominoes,* Congr. Numer. **145** (2000), 65–80.

[HW] F. Harary, M. Weisbach, *Polycube achievement games,* J. Recreational Mathematics **15** (1982-83), 241–246.

[HS1] H. Harborth, M Seemann, *Handicap achievement for squares,* To appear.

[HS2] H. Harborth, M Seemann, *Snaky is an edge-to-edge loser,* Geombinatorics **5** (1996), no. 4, 132–136.

[HS3] H. Harborth, M Seemann, *Snaky is a paving winner,* Bull. Inst. Combin. Appl. **19** (1997), 71–78.

[Plu] A. Pluhár, *Generalized Harary games,* Acta Cybernet. **13** (1997), no. 1, 77–83.

[Sie] N. Sieben, *Wild polyomino weak* $(1,2)$*-achievement games,* preprint.

Received June, 2004

Acta Cybernetica **16** (2004) 587–600.

GeLexi project: Sentence Parsing Based on a GEnerative LEXIcon*

Gábor Alberti,* Judit Kleiber,* and Anita Viszket†

Abstract

The principal aim of our research team[1], called GeLexi, is to legitimate a new sort of generative grammar via verifying its computational implementability. This grammar is more radically "lexicalist" than any earlier one: no phrase structure trees are generated, but word order is accounted for by means of ranked parameters. Another novelty is the extension of "total lexicalism" to morphology: lexical items are assigned not to words but to morphemes. Our parser, in accordance with the basic task of every generative grammar, decides whether a sentence is grammatical, and if it is, then provides a morphophonological analysis, a compilation of grammatical relations, and two kinds of semantic representations. At the end we show some examples to demonstrate our procedures, among them a sentence containing the conjunction *és* 'and', which is our latest development.

1 Introduction

Our general aim is to verify that computational linguistics is worth returning from the nowadays wide-spread attitude characterized by "shallow parsing" (which is held to save expenses) to the pure theoretical (generative) linguistic basis. A formal (generative) grammar can be elaborated [4] showing the distribution of capacity advantageous in modern computer science: "minimal processing - maximum database".[2]

The starting aim of our team (section 2) is the same, approaching from a theoretical point of view: we would like to legitimate a new sort of generative grammar

*Presented at the 1st Conference on Hungarian Computational Linguistics, December 10–11, 2003, Szeged.

†University of Pécs, Faculty of Humanities, Linguistics Department, H7624 Pécs, Ifjúság útja 6., Hungary, e-mail: gelexi@btk.pte.hu

[1]We are grateful to the Hungarian National Scientific Research Found (OTKA T038386). Further thanks to Kata Balogh, who used to be an active member of the team.

[2]In harmony with the chance available now: to use a significantly greater number of huge patterns than earlier due to the immense increase in memory capacity. In the meantime generative linguistics, which used to be chiefly "process-oriented" (i.e. syntax-centered) in its first period [16], took a sweeping lexicalist turn [14, 15, 17, 19, 20]. The current attitude can be characterized by two mottoes of Joshi's [19], the father of mildly context-sensitive grammars [22]: "Complicate Locally, Simplify Globally", and "Grammar ≈ Lexicon".

(GASG: *Generative Argument Structure Grammar*) by working out its computational implementation (because a successful implementation is the best evidence for the exactness and consistency of a formal system). This grammar is more radically lexicalist than any earlier one, since no phrase structure trees are built, yet word order can be accounted for by means of ranked parameters for requirements concerning *immediate precedence* relations between words.

In section 3 our four-year work is looked over according to the assumption that the principal aim of the first national conference on computational linguistics is the introduction of the teams working on this field.

Section 4 is denoted to the demonstration of some examples in order to elucidate how our parser works, concentrating on the conjunction *és* 'and' in the spirit of the strategy that at all conferences we present a new achievement beyond summarizing our general ideas.

2 Starting aim

Due to generative linguistics [16], there is a formal frame to express the old recognition that the meaning of a sentence comes from two sources: lexical items and the structure they form. Initially the main question was studying the combinatorial possibilities of these structures by mathematical means (see the Chomskyan hierarchy of grammars, especially context free and context sensitive grammars [16] [22]). From these studies two fields started developing: generative language description and computational linguistics. In the generative paradigm the indispensability of transformation rules (moving constituents) could not be theoretically proved [22], consequently from the 70s several new (generative) approaches could be established [14] [15] [18] [19] which dispense with transformation (Partee et al [22] show all the variants of these *"mildly" context free* grammars). In connection with this, in these new approaches the *lexicon* plays the crucial role (instead of syntax, as earlier) in describing the linguistic phenomena, and there are only highly general phrase structure rules in *syntax*. The fact that in the 90s even the previously strictly transformational Chomskyan "main line" took a similar turn (the Minimalist Program [17]) shows that the lexicalist tendency is extremely strong nowadays.

GASG is a totally lexicalist grammar, which accomplishes Karttunen's "radical lexicalism" [20]: it is a grammar which dispenses with not only transformational rules but phrase structure trees as well (it is similar to dependency grammars [24] in this respect); yet it fulfills the basic generative task, namely that the set of grammatical sentences can be defined, and (through the analysis) a syntactic and semantic representation can be assigned to the sentence in question. Grammatical knowledge is built only in the descriptions of lexical items: each morpheme declares what kind of "environmental requirements" a sentence containing the given lexical item has to fulfill. Obviously, this is not the only information that these lexical units have: their own features are also registered in these descriptions, since other morphemes need this information when they "intend" to form sentences with them.

Generative linguistic theories are important to be verified by computational im-

plementations, for having working algorithms is the best evidence for the exactness and consistency of a formal system. Our implementation does fulfill the basic generative task: it decides whether the input sentence is grammatical or not, and if it is, then it provides a syntactic and semantic representation.

We will turn back to other properties of our grammar and parser later on, but now let us point out a further advantage of this (lexicalist) approach. These days a significantly greater number of huge patterns can be used than earlier due to the immense increase in memory capacity [23]. It can be favorable to combine this technical development with the lexicalist approach, especially with a homogeneous grammar. Further on, GASG can legitimate the heuristic procedures (based on pattern matching), which were previously used intuitively, by improving them to theoretically correct systems.

3 Previous work

The idea of developing a computational implementation based on GASG was first published in 1998 at an international conference in Debrecen [1]. The first version of our parser was done in 2001 which could decide whether a Hungarian sentence was grammatical or not [10, 3]. It could account for regent-argument relations, agreement, and word order, but we had neither semantic nor morphological component: lexical items were assigned to words, not morphemes like now.

Then we started elaborating the semantic representation of GASG. The theory serving as the starting point was a developed version [2] of Kamp's DRT (Discourse Representation Theory) [18]. The main argument for choosing this approach was that GASG could be the *compositional* [22] grammar for this discourse semantic theory, which is much more advanced than the Montagovian semantic systems [22]. Our concepts until then were demonstrated in details in the Proceedings of a conference about principles, models, and rules published in Szeged [4]. Morphology is not discussed there, because until then lexical items were (morphologically complex) words, and were claimed to be arranged in a multiple lexical inheritance network put together by means of well-tried regular devices [21].

Later it became evident that for achieving the required complete homogeneity we should regard morphology in a totally lexicalist way as well. Therefore lexical items are not associated with words but with morphemes (stems and affixes) in this new approach. It is also important that these lexical items have all the information needed on each linguistic level to form sentences, while it is also decided which morphemes are put together, and which morphemes form independent words [5, 11, 12], which is not (necessarily) the same in different languages, e.g. in Hungarian *olvas-hat* 'read-CAN' (one word), but in English *can read* (two independent words). Another fundamental point of GASG and our parser is what technique the phrase structure (which is responsible for word order) is replaced with, namely checking the ranked *immediate precedence* requirements the lexical items have in their descriptions. These requirements can be fulfilled directly (by being next to the given morpheme) or indirectly, when requirements with higher ranks are met.

This is how between words belonging together (e.g. *the girl*) other words can be inserted (*the proud Hungarian girl*), in Hungarian even with their further dependants (*a két fiára büszke magyar lány* – 'the *two of her sons* proud Hungarian girl'. In English in this case the adjective has to be after the noun: *the Hungarian girl proud of her two boys*). Morphotax can also be entrusted to rank parameters with the important difference that morphemes cannot bring further dependants (which correlates with the regularity of morphology [21]).

The idea of a totally lexicalist morphology can make the differences between language types irrelevant on the abstract level of *copredicative network* [8], since it does not matter whether the (English) words *I may wait for you* or the (Hungarian) morphemes *vár-hat-l-ak* 'wait-CAN-2SGobj-1SGsubj' are looking for each other. We have started developing a machine translating system through this level [7]. The first results were demonstrated at the 2004 (Maltese) workshop of the European Association for Machine Translation (EAMT'04) [9].

The morpheme-based semantic representation was introduced at a conference in Mexico [6], where we also took the opportunity to publish the mathematical definition of GASG grammar type.

4 The present parser

In what follows, we introduce the components of our parser from morphophonology until semantics presenting some examples as well, first to show how the components work, and then to demonstrate how the conjunction *és* 'and' has been worked out.

4.1 Morphophonology

The input of this level (and the whole parser as well) is a simple string, a series of words. The first task the program has to fulfill is *segmentation.* The relevant lexical items are to be identified on the basis of the database, which has been found in the same program so far, but we are planning to use a relational database instead, for storing lexical items, which we have already started developing[3].

Lexical items consist of two parts, the *own word* and a label. The former shows how the lexical items appear in different environments. Sometimes it is only one form which is possible (e.g. *Mari*, meaning *Mary*), but sometimes more than one variants exist (e.g. *bokor* or *bokr-*, meaning *bush*; *-t/-at/-et/-ot/-t* for accusative case). In the latter case variables are used (*bokOr*, *-Vt*). The different features of the lexical items are stored in their label (first the phonological, then morphological and syntactic properties), but first the English "name" of the predicate is stored[4].

[3]The members of the team working on this project (called *LiLe project*, where 'LiLe' stands for Linguistic Lexicon) are Anita Viszket, Éva Szilágyi, Zoltán Bódis and Judit Kleiber (University of Pécs, Linguistics Department).

[4]It is often asked why we do not use existing encoding systems or even existing morphological parsers. The answer is that we need much more (and sometimes quite different) information than these systems store, because we aim at producing more detailed analyses.

After identifying the relevant lexical items the program checks the well-formedness of the words (it can account for such linguistic phenomena as vowel harmony, shortening, lenghtening, epenthesis, lowering, etc.), and the morpheme order. The output of this component is a list containing these lexical items. If there are more than one possible segmentation of a word, the program can print out each of them. Let us demonstrate this through an example.

```
grammword("dobom."),fail.

dob: n(1,1,li(m("","dob",""),labstem("throw",phonfst(2,1,2,2),2,[["NOM","ACC"]])))
om: n(1,2,li(m("V","m",""),labsuff("sg1obj+def",phonfsu(1,1,1,3),2,3)))

dob: n(1,1,li(m("","dob",""),labstem("drum",phonfst(2,1,2,2),1,[])))
om: n(1,2,li(m("V","m",""),labsuff("possI",phonfsu(1,1,1,1),1,2)))
```

Asking the well-formedness of the word *dobom*, the program provides two solutions, a verb (*throw*) in 1SG definite conjunction, and a noun (*drum*) with the possessive 1SG. So we can account for ambiguities on this linguistic level.

Finally we quote the morphological output of the parser for a more complex sentence. At the beginning of each line the proper morph can be seen, and the numbers after that (and before the own word) mean that the given morpheme is (1) in which word and (2) which morpheme within that word.

```
gramm("Péter keresteti Marit a magyar rendörséggel.").
Peter look-for - cause - 3sg Mary-Acc the Hungarian police-Instr
'Peter makes the police look for Mary.'

LEXICAL ITEMS:
Péter: n(1,1,li(m("","Péter",""),labstem("Peter",phonfst(1,2,0,2),1,[])))
keres:
n(2,1,li(m("","keres",""),labstem("look-for",phonfst(1,2,2,2),2,[["NOM","ACC"]])))
tet: n(2,2,li(m("t","A","t"),labder("cause",phonfsu(2,2,0.2,2),2,ac(-1,0,1))))
i: n(2,3,li(m ("","i",""),labsuff("sg3obj+def",phonfsu(1,3,1,3),2,3)))
Mari: n(3,1,li(m("","Mari",""),labstem("Mary",phonfst(2,2,0,2),1,[])))
t: n(3,2,li(m("V","t",""),labsuff("ACC",phonfsu(1,1,1,3),1,4)))
a: n(4,1,li(m("","a","Z"),labstem("the",phonfst(1,3,3,3),3,[])))
magyar: n(5,1,li(m("","magyar",""),labstem("Hungarian",phonfst(2,2,1,1),4,[])))
rendörség: n(6,1,li(m("","rendörség",""),labstem("police",phonfst(1,2,3,2),1,[])))
gel: n(6,2,li(m("S","A","l"),labsuff("INSTR",phonfsu(1,2,2,3),1,4)))
```

4.2 Syntax

The input of this level is the list of numbered lexical items (see above). The task this component has to fulfill is to satisfy all the requirements these lexical units have in their descriptions concerning syntax.

The most important requirements are regent (predicate) - argument relations. The verb *vár* 'wait', for instance, has two argument structures in the program (Nom+Acc; Nom+Sublat), so the parser has to find two nouns with these case marking morphemes (the first two or the second two) in order to fulfill one of its requirements. This searching has to be mutual, so these nouns have to find the same verb when they are looking for their regent. This is to exclude sentences like **Péter*

vár Marira Julira *'Peter waits for Mary Julie.', where two nouns in sublative case are looking for the same verb, so one of their searching cannot be mutual. There is another type of relation (free relations) where the seeking is one-way, e.g. the relation between an adjective and a noun. The adjective needs a noun, but not the other way round; that is why a noun can have more adjectives.

After finding these (mutual or one-way) relations the program checks word order by means of ranked *immediate precedence* requirements, which we have already mentioned. With this technique we can also explain, for example, why free adverbs and arguments can be freely mixed in Hungarian but not in English: suppose regent-argument relation has an *immprec* rank α, free adverb-verb relation has a rank β, and in Hungarian $\alpha = \beta$, but in English $\alpha < \beta$.

If the parser can fulfill all the requirements, the syntactic relations are printed out in two different formats, the first one is more detailed than the much simplified second one. Let us demonstrate this through the previous example.

```
1. gramm("Péter keresteti Marit a magyar rendörséggel.").
2. SYNTAX:
3. gr("noun","regent","subj",1,1,2,1)
4. gr("det","regent","_",1,1,2,1)
5. gr("regent","noun","subj",2,1,1,1)
6. gr("regent","det","subj",2,1,1,1)
7. gr("regent","noun","obj",2,1,3,2)
8. gr("regent","det","obj",2,1,3,1)
9. gr("suff","stem","free",2,2,2,1)
10. gr("regent","noun","obl",2,2,6,2)
11. gr("regent","det","obl",2,2,4,1)
12. gr("suff","stem","free",2,3,2,1)
13. gr("det","regent","_",3,1,2,1)
14. gr("suff","stem","free",3,2,3,1)
15. gr("noun","regent","obj",3,2,2,1)
16. gr("det","noun","free",4,1,6,1)
17. gr("det","regent","_",4,1,2,1)
18. gr("adj","noun","free",5,1,6,1)
19. gr("suff","stem","free",6,2,6,1)
20. gr("noun","regent","obl",6,2,2,1)
21. regent-noun-subj: keresteti-Péter
22. regent-det-subj: keresteti-Péter
23. regent-noun-obj: keresteti-Marit
24. regent-det-obj: keresteti-Marit
25. regent-noun-obl: keresteti-rendörséggel
26. regent-det-obl: keresteti-a
27. det-noun: a-rendörséggel
28. adj-noun: magyar-rendörséggel
```

In the first part three types of free relations and several types of regent-argument relations can be distinguished. First the names of the relations can be seen, and then the numberings of the morphemes which establish the given relation. Free relation types are *suff-stem-free*, when a suffix is in relation with its stem (lines 9, 12, 14, 19), *det-noun-free*, when a determiner found its noun (line 16), and *adj-noun-free*, when an adjective is in relation with its noun (line 18). The regent-argument relation types are *regent-noun-subj/obj/obl*, when a predicate found one

of its nominal arguments (lines 5, 7, 10), *regent-subj/obj/obl*, when the regent is in relation with a determinative element, because we claim that the arguments have two pillars, a noun and a determiner[5] (lines 6, 8, 11), or – because regent-argument relations are mutual – *noun-regent-subj/obj/obl*, when the arguments found their regents (lines 3, 15, 20), and *det-regent-_*, when the determiner pillar of the argument is in relation with the regent (lines 4, 13, 17), where '_' stands for either *subj*, *obj* or *obl* (it cannot be decided, because – in Hungarian – there are no markings on the determiner pillar concerning syntactic roles). In the second part the same information can be found, but mutual relations appear only once, and the numberings are replaced by the words themselves, so it can be read much easier than the first representation.

It is an interesting point of our approach that not necessarily words (stems) look for each other, but affixes as well. E.g. in the sentence *Énekeltetem Marit* (*sing-CAUSE-1SG Mary-ACC*) 'I make Mary sing' the verb itself do not even require a (human) object, consequently it cannot be the verb stem that searches for the object. The morpheme which needs (thus legitimates) the object in this sentence is the causative morpheme (derivative suffix) *-tAt*, and the morpheme it looks for is not the whole *Marit* or the stem *Mari*, but the accusative case marking suffix *-Vt*, because it is this element that shows that *Marit* is an object in this sentence.

4.3 Semantics

The input of this level is all the information we have at this point (list of lexical items, syntactic relations). The task this component has to fulfill is to provide a DRS (Discourse Representation Structure) putting it together from the proto-DRSs the lexical items give.

The information necessary for producing these proto-DRSs can be found in the descriptions of lexical items. Determiners provide referents (*provref*), and predicates make statements (*pred*). In the sentence *The boy loves a girl*, say, there are two referents *r1* and *r4* (the first and the fourth words provide them), and three statements, that *boy(r1)*, *girl(r4)*, and *love(r1,r4)*. Further on, it is also known that *r1* is an old referent (it has been provided earlier), but *r4* is new. This is the point where different kinds of filters can be built in, e.g. that the first argument of the predicate *love* must be human.

If the program can produce the proto-DRSs and put them together to a final DRS, it is printed out. Let us turn back to the previous example again, and see the discourse semantic output of the parser when the grammaticality of the sentence *Péter keresteti Marit a magyar rendőrséggel* (*Peter look-for-CAUSE-3SG Mary-ACC the Hungarian police-INSTR*) 'Peter makes the Hungarian police look for Mary' is asked.

[5]Sometimes this nominal pillar and the determiner pillar are within one morpheme, e.g. in the case of proper names; and there are also exceptions, e.g. *Péter moziba megy* (*Peter cinema-ILLAT go-3SG*) 'Peter goes to the cinema', which requires a different approach.

```
1. gramm("Péter keresteti Marit a magyar rendörséggel.").
2. DISCOURSE SEMANTICS:
3. provref("fixpoint",[e(2,2,1)])
4. provref("old",[r(1,1,1)])
5. pred("Peter",1,[r(1,1,1)])
6. provref("new",[e(2,1,1)])
7. pred("look-for",2,[e(2,1,1),r(4,1,1),r(3,1,1)])
8. provref("new",[e(2,2,1)])
9. provref("=",[e(2,2,1),e(2,1,1)])
10. pred("cause",2,[e(2,2,1),r(1,1,1),e(2,1,1)])
11. provref("old",[r(3,1,1)])
12. pred("Mary",3,[r(3,1,1)])
13. provref("old",[r(4,1,1)])
14. provref("<or=",[r(4,1,1),e(2,2,1)])
15. pred("Hungarian",5,[r(4,1,1)])
16. pred("police",6,[r(4,1,1)])
```

Referents are provided for Peter (line 4), Mary (11), and the police (13), each of them are old referents, and it is predicated that *r111* is Peter (5), *r311* is Mary (12), and *r411* is the police (16), which is Hungarian (15). In this approach other kinds of referents are provided, such as Davidsonian (eventuality) referents for the events (lines 6, 8), where *e211* is that *r411* is looking for *r311* (line 7), and that *r111* causes the event *e211* (line 10)[6].

In comparison with earlier approaches there are two important changes in this representation. First, in this version, *morphemes* provide the proto-DRSs, so that we can give a more precise interpretation, and, second, we try to give a formulation, which easily fits in the structure of a Lifelong DRS [2], which goes beyond sentence parsing. This method could enable our program to parse whole texts, not only sentences. LDRT provides a partial ordering between possible worlds, this ordering can be seen in lines 9 and 14.

Hungarian is a pro-drop language, which means that personal pronouns do not have to appear in the sentence. Several approaches assume empty elements in this case, but we do not. In GASG (and our parser) in a sentence like *Szeretlek.* 'I love you.' syntax is very simple, it consists of only *suff-stem-free* relations. What is more interesting is semantics, where we have to account for the first and the second argument of the predicate. In our approach if (pro)nouns are not present in the sentence, then the conjugation (agreement suffixes on the verb) will be responsible for showing the subject and the object.

```
gramm("Szeretlek.").
LEXICAL ITEMS:
szeret: n(1,1,li(m("","szeret",""),
        labstem("love",phonfst(1,2,2,2),2,[["NOM","ACC"]])))
l: n(1,2,li(m("","l",""),labsuff("objperson2",phonfsu(3,2,1,1),2,2.5)))
ek: n(1,3,li(m("V","k",""),labsuff("sg1",phonfsu(1,1,2,3),2,3)))
```

[6] If we have a similar sentence, but without a noun in instrumental case, the first argument of the predicate *'look-for'* will be an unknown referent *r000.*

```
SYNTAX:
gr("suff","stem","free",1,2,1,1)
gr("suff","stem","free",1,3,1,1)
DISCOURSE SEMANTICS:
provref("fixpoint",[e(1,1,1)])
provref("new",[e(1,1,1)])
pred("love",1,[e(1,1,1),r(0,1,1),r(0,1,2)])
```

It can be seen that the information that who loves is *me* (*r011*, where *0* means that this is an inbuilt referent, and *11* means singular, first person), and who is loved is you-singular (*r012*), and of course this knowledge could not come from anywhere else than from the two suffixes of the verb.

4.4 Copredicative Network

This component does not belong strictly to the parsing mechanism but is an abstract level between syntax and semantics showing which propositions "co-predicate" and how, and is useful in (machine) translation, because it preserves something from the original structure of the sentence, but the predicate-argument relations also appear in it.

The input of this level is the list of the relevant lexical items and the syntactic relations, and the output is the list of copredications. Let us show it through the familiar example.

```
1. gramm("Péter keresteti Marit a magyar rendörséggel.").
2. COPREDICATIVE NETWORK:
3. copr("look-for",2,1,"Peter",1,1,1,1,"arg")
4. copr("look-for",2,1,"Peter",1,1,1,0,"arg")
5. copr("look-for",2,1,"Mary",3,1,2,1,"arg")
6. copr("look-for",2,1,"Mary",3,1,2,0,"arg")
7. copr("cause",2,2,"Peter",1,1,1,1,"arg")
8. copr("cause",2,2,"Peter",1,1,1,0,"arg")
9. copr("cause",2,2,"look-for",2,1,2,0,"arg")
10. copr("the",4,1,"police",6,1,0,1,"free")
11. copr("Hungarian",5,1,"police",6,1,1,1,"free")
```

In line 3 it can be seen that there is a copredication between the predicate *look-for* (second word, first morpheme) and the argument *Peter* (first word, first morpheme), namely that the first argument (1) of the former is the first argument (1) of the latter (which is, in the case of arguments means that *itself*). There is another first argument of this predicate (line 4), which is the zeroth argument of *Peter*, namely its determiner pillar (it has already been mentioned that predicates look for their arguments on two pillars). The same can be found in lines 5-6, now with the second argument of the predicate. Lines 7-8 show that it is Peter that causes something, which is in line 9, and the second argument of the predicate *cause* is the zeroth argument of *look-for*, namely the event itself. These all have been regent-argument relations (*"arg"*), and there are two free-relations in lines 10-11.

4.5 Conjunction

Up to this point we have introduced our parser in general, and now we show our latest development, *conjunction*. First we worked out the case when two nouns are conjoined, but we claim that other types of conjunctions can be treated similarly.

To show how this works, we finally quote the whole output of our parser, when the grammaticality of the input string (1) is questioned, in which a conjunction (*és* 'and') can be found.

```
1. Mari és Juli kereshetik Pétert.
   Mary and Julie look-for - may - 3pl Peter-Acc
   'Mary and Julie may look for Peter.'

2. LEXICAL ITEMS:
3. Mari: n(1,1,li(m("","Mari",""),labstem("Mary",phonfst(2,2,0,2),1,[])))
4. és: n(2,1,li(m("","s",""),labstem("and",phonfsu(1,1,1,1),5,[])))
5. Juli: n(3,1,li(m("","Juli",""),labstem("Julie",phonfst(2,2,0,2),1,[])))
6. keres: n(4,1,li(m("","keres",""),
           labstem("look-for",phonfst(1,2,2,2),2,[["NOM","ACC"]])))
7. het: n(4,2,li(m("h","A","t"),labsuff("may",phonfsu(1,1,1,2),2,1)))
8. ik: n(4,3,li(m("","ik",""),labsuff("pl3def",phonfsu(2,3,1,3),2,3)))
9. Péter: n(5,1,li(m("","Pter",""),labstem("Peter",phonfst(1,2,0,2),1,[])))
10. t: n(5,2,li(m("V","t",""),labsuff("ACC",phonfsu(1,1,1,3),1,4)))

11. SYNTAX:
12. gr("noun","regent","conj",1,1,2,1)
13. gr("det","regent","conj",1,1,2,1)
14. gr("regent","noun","conj",2,1,1,1)
15. gr("regent","det","conj",2,1,1,1)
16. gr("noun","regent","subj",2,1,4,1)
17. gr("regent","noun","conj",2,1,3,1)
18. gr("regent","det","conj",2,1,3,1)
19. gr("det","regent","subj",2,1,4,1)
20. gr("noun","regent","conj",3,1,2,1)
21. gr("det","regent","conj",3,1,2,1)
22. gr("regent","noun","subj",4,1,2,1)
23. gr("regent","det","subj",4,1,2,1)
24. gr("regent","noun","obj",4,1,5,2)
25. gr("regent","det","obj",4,1,5,1)
26. gr("suff","stem","free",4,2,4,1)
27. gr("suff","stem","free",4,3,4,1)
28. gr("det","regent","__",5,1,4,1)
29. gr("suff","stem","free",5,2,5,1)
30. gr("noun","regent","obj",5,2,4,1)
31. regent-noun-conj: és-Mari
32. regent-det-conj: és-Mari
33. regent-noun-conj: és-Juli
34. regent-det-conj: és-Juli
35. regent-noun-subj: kereshetik-és
36. regent-det-subj: kereshetik-és
37. regent-noun-obj: kereshetik-Pétert
38. regent-det-obj: kereshetik-Pétert

39. COPREDICATIVE NETWORK:
40. copr("and",2,1,"Mary",1,1,1,0,"arg")
41. copr("and",2,1,"Mary",1,1,1,1,"arg")
42. copr("and",2,1,"Julie",3,1,2,0,"arg")
43. copr("and",2,1,"Julie",3,1,2,1,"arg")
```

```
44. copr("look-for",4,1,"and",2,1,1,1,"arg")
45. copr("look-for",4,1,"and",2,1,1,0,"arg")
46. copr("look-for",4,1,"Peter",5,1,2,1,"arg")
47. copr("look-for",4,1,"Peter",5,1,2,0,"arg")
48. copr("may",4,2,"look-for",4,1,2,0,"arg")
49. SEMANTICS:
50. provref("fixpoint",[e(4,2,1)])
51. provref("old",[r(1,1,1)])
52. pred("Mary",1,[r(1,1,1)])
53. provref("new",[r(2,1,1)])
54. provref("<or=",[r(2,1,1),e(4,1,1)])
55. pred("element",2,[r(1,1,1),r(2,1,1)])
56. pred("element",2,[r(3,1,1),r(2,1,1)])
57. provref("old",[r(3,1,1)])
58. pred("Julie",3,[r(3,1,1)])
59. provref("new",[e(4,1,1)])
60. pred("look-for",4,[e(4,1,1),r(2,1,1),r(5,1,1)])
61. provref("new",[e(4,2,1)])
62. provref("<",[e(4,2,1),e(4,1,1)])
63. pred("may",4,[e(4,2,1),e(4,1,1)])
64. provref("old",[r(5,1,1)])
65. pred("Peter",5,[r(5,1,1)])
66. yes
```

To start at the end, line 66 shows that the sentence is grammatical, which means that the program could identify the relevant lexical items (stems and affixes, lines 2-10) in an appropriate morphophonological environment [5], and the relevant syntactic relations, which can be seen in lines 11-38. Looking at the simplified representation (31-38), it can be read that there is a relation between the conjunction and Mary (31-32), and between the conjunction and Julie (33-34), and this conjunction is the one that establishes relation with the verb (lines 35-36). Lines 37-38 show that the object of the verb is Peter. The same is expressed in lines 39-48 but in a more abstract way, the copredicative network shows the relations between the (form-independent) semantic units.

Discourse semantics can be seen in lines 49-65. The main points are that a situation (*e411*) is probable (line 63), and this situation is that *r211* is looking for *r511* (60), where *r211* is a group consisting of *r111* (55) and *r311* (56), *r111* is Mary (52), *r311* is Julie (58), and *r511* is Peter (65).

In the case of verbs with *groups* as subjects, there arises a problem of agreement. To solve it, we have (partially) adopted Bánréti's suggestion [13] for our parser, namely that in these cases plural conjugation must be used with the "highest" ranked person ($1 > 2 > 3$) among all the co-ordinated nouns' persons. For example in the sentences *Én és te/ti/Péter kereshetjük Marit* (*I and you$_{sg}$/you$_{pl}$/Peter look-for-CAN-1PLdef*) 'I and you$_{sg}$/you$_{pl}$/Peter may look for Mary', the verb is in 1PL.

Another interesting problem is to interpret collective actions. The approach behind the present version of the parser is that collective reading illustrated above is generally sufficient in the semantic representation: it can be decided later if we need the referents one by one. There is only one case in which we assume a distributive reading, when the verb and the co-ordinated subjects are all in 3SG, for example in the sentence *Péter és János keresi Marit* (*'Peter and John*

look-for-3SGdef Mary-ACC) 'Peter and John are looking for Mary'. In this case Peter and John are looking for Mary *separately*, while in the former example (with a verb in plural) they were doing it together. The semantic representation can show the difference.

```
1. DISCOURSE SEMANTICS:
2. provref("fixpoint",[e(4,1,1)])
3. provref("old",[r(1,1,1)])
4. pred("Peter",1,[r(1,1,1)])
5. provref("new",[r(2,1,1)])
6. provref("<or=",[r(2,1,1),e(4,1,1)])
7. pred("element",2,[r(1,1,1),r(2,1,1)])
8. pred("element",2,[r(3,1,1),r(2,1,1)])
9. provref("old",[r(3,1,1)])
10. pred("John",3,[r(3,1,1)])
11. provref("new",[e(4,1,1)])
12. pred("look-for",4,[e(4,1,1),r(1,1,1),r(5,1,1)])
13. provref("new",[e(4,1,2)])
14. pred("look-for",4,[e(4,1,2),r(3,1,1),r(5,1,1)])
15. provref("old",[r(5,1,1)])
16. pred("Mary",5,[r(5,1,1)])
```

Compared this distributive interpretation to the previous one, we can see that two actions are concerned instead of one (lines 11-14), the two subject are looking for *r511* separately. Nevertheless, of course, they form a group in this case as well (lines 7-8), because we can refer to their common referent.

5 Conclusion and future work

We hope that even this brief demonstration could prove that our program can provide much wider parsing than usual on a non-trivial lingustic fragment. We are planning to enlarge this fragment, extend our system to other languages, and develop further "intelligent" applications[7]. The guarantee for realizability is the theoretically "pure" and practically simplified processing. Growing out of the experimental phase, we are about to switch over to a more efficient programming language and query analizer, and we are seeking the opportunity to use some kind of corpus to simulate knowledge bases.

References

[1] Alberti, G.: GASG: Minimal Syntax, Maximal Lexicon and PROLOG. Paper read at ALLC/ACH '98, July 9. In Hunyadi, L. (ed.): ALLC/ACH '98. KLTE, Debrecen (1998) 81–83.

[7]Since the time of the conference (MSZNY2003), we have worked out the theoretical basis of machine translation, and now our parser can translate a particular group of sentences from Hungarian into English and vice versa (EAMT'04).

[2] Alberti, G.: Lifelong Discourse Representation Structures. Gothenburg Papers in Computational Linguistics 00-5 (2000) 13–20.

[3] Alberti, G., Balogh, K., Kleiber, J.: GeLexi Project: Prolog Implementation of a Totally Lexicalist Grammar. In de Jongh, Zeevat, Nilsenova (eds.): Proc. of the Third and Fourth Tbilisi Symp. on Language, Logic and Computation. ILLC, Amsterdam, and Univ. Tbilisi (2002).

[4] Alberti, G., Balogh, K., Kleiber, J.,Viszket, A.: A totális lexikalizmus elve és a GASG nyelvtan-modell [The Principle of Total Lexicalism and the GASG grammar model]. Maleczki, M. (szerk.): A mai magyar nyelv leírásának újabb módszerei V. Szegedi Tudományegyetem (2002) 193–218.

[5] Alberti, G., Balogh, K., Kleiber, J.,Viszket, A.: Towards a totally lexicalist morphology. Talk at 6th International Conference on the Structure of Hungarian (ICSH6), Düsseldorf, Germany (2002). To appear in Kenesei, I., Piñón, Ch. (eds.): Approaches to Hungarian 9.

[6] Alberti, G., Balogh, K., Kleiber, J.,Viszket, A.: Total Lexicalism and GAS-Grammars: A Direct Way to Semantics. In Gelbukh, A. (ed.): Proceedings of CICLing2003 (Mexico City). LNCS N2588. Springer-Verlag, Berlin Heidelberg New York (2003) 37–48.

[7] Alberti, G., Balogh, K., Kleiber, J.,Viszket, A.: A fordítás totálisan lexikalista megközelítése [The Totally Lexicalist Approach of Machine Translation]. MANYE, Számítógépes nyelvészeti szekció, Győr (2003).

[8] Alberti, G., Kleiber, J.: Extraction of Discourse-Semantic Information... In Cunningham, H., Paskaleva, E., Bontcheva, K., Angelova, G. (eds.): Information Extraction for Slavonic and Other Central and Eastern European Languages. Borovets, Bulgaria (2003) 63–69.

[9] Alberti, G., Kleiber, J.: The GeLexi MT Project. In J. Hutchins (ed.): Proceedings of EAMT 2004 Workshop, Valletta: Univ. of Malta (2004) 1–10.

[10] Balogh, K., Kleiber, J.: Egy lexikalista nyelvtan morfoszintaxisának PROLOG-implementációja [The PROLOG-implementation of the morphosyntax of a Totally Lexicalist Grammar]. JGYTF, Szeged (2001).

[11] Balogh, K., Kleiber, J.: Computational Benefits of a Totally Lexicalist Grammar. In Matouek, V., Mautner, P. (eds.): Text, Speech and Dialogue, Proceedings of TSD2003. Springer-Verlag, Berlin Heidelberg New York (2003) 114–119.

[12] Balogh, K., Kleiber, J.: A Morphology Driven Parser for Hungarian. Talk at the 5th Int. Tbilisi Symp. on Language, Logic and Computation. Org. by ILLC, Amsterdam, and U. Tbilisi (2003).

[13] Bánréti, Z.: A mellérendelés [Conjunction]. Kiefer, F. (szerk.) Strukturális magyar nyelvtan I. Mondattan. Akadémiai, Budapest (1992) 715–796.

[14] Borsley, R. D.: Modern Phrase Structure Grammar. Blackwell, Oxford Cambridge (1996).

[15] Bresnan, J.: Lexical Functional Syntax. Blackwell, Oxford (2000).

[16] Chomsky, N.: Syntactic Structures. The Hague, Mouton (1957).

[17] Chomsky, N. (ed.): The Minimalist Program. MIT Press, Cambridge, Mass. (1995).

[18] van Eijck, J., Kamp, H.: Representing discourse in context. In van Benthem, J., ter Meulen, A. (eds.): Handbook of Logic and Language. Elsevier, Amsterdam, The MIT Press, Cambridge, Mass. (1997).

[19] Joshi, A. K.: Starting with Complex Primitives Pays Off. In Gelbukh, A. (ed.): Proceedings of CICLing2003 (Mexico City). LNCS N2588. Springer-Verlag (2003) 1–10.

[20] Karttunen, L.: Radical Lexicalism. Report No. CSLI 86 68, Stanford (1986).

[21] Karttunen, L.: Computing with Realizational Morphology In Gelbukh, A. (ed.): Proceedings of CICLing2003 (Mexico City). LNCS N2588. Springer-Verlag (2003) 203–214.

[22] Partee, B., ter Meulen, G.B., Wall, R.P.: Mathematical Methods in Linguistics. Kluwer Academic Publ. (1990).

[23] Prószéky, G.: Megértéstámogatás és gépi fordítás: nyelvtechnológia a XXI. század elején [Text Understanding and Machine Translation: Language Technology in the Early Years of the 21st Century]. Eighth National Neumann Congress, Hungary (2003).

[24] Schubert, K.: Metataxis (Contractive Dependency Syntax for Machine Translation). Foris, Dordrecht (1987).

Received May, 2004

Acta Cybernetica 16 (2004) 601–609.

A Semantic Network Dictionary for Dysphasia Therapy*

János Bácsi†

Abstract

With the use of computer-assisted teaching and learning programs the speech therapy of children with dysphasia can be planned more effectively, than with the traditional, so-called prompted-image method employed by logopaedists. Our aim is the creation of a semantic network dictionary – on the basis of a huge quantity empirical database –, which predicts what other concepts will link to a concept in the STM of 4—7 year old children, i.e. what other concepts can the most easily be associated with an already existing one by the therapeut. Furthermore, the semantic network dictionary will enable the already literate child – on the basis of the idea of András Kocsor – to take a virtual tour through the connected words of the network dictionary by merely saying words into a microphone. Hence the software will help children develop and maintain semantic structures.

1 Introduction

The cognitive sciences, memory research, research of artificial intelligence, modern linguistics, computer linguistics, evolution psychology and modern learning theories have arrived at such results, which instigated therapist dealing with children with dysphasia, and researchers dealing with acquisition of written language to re-evaluate and reconsider their methods employed so far.

The development of children suffering from expressive language disorder cannot be overlooked, as language development with dysphasia is not language-specific, in all the languages of the world – where such survey has been performed – 4—7 % of the given population (4—8 year old children) is affected. The affected children must be taught separately, because without special development, acquisition of visual language (writing, reading) will cause problems. A child with dysphasia in the process of institutionalised education will show symptoms of dyslexia and/or disgraphy, which is not taken up at school like an infection, but as a result of expressive language deficit, the inappropriate level of necessary capabilities to acquire the visual language will surface it.

*Presented at the 1st Conference on Hungarian Computational Linguistics, December 10–11, 2003, Szeged.

†Juhász Gyula Teacher Training College Division, Teacher Training School, University of Szeged, H-6725 Szeged, Boldogasszony sgt. 8, Hungary, e-mail: bacsi@jgytf.u-szeged.hu

The most important characteristic feature of expressive disorder is limited vocabulary (see DSM-IV), hence the prime task of the therapist planning and maintaining development is the extension of vocabulary. This task is usually undertaken by logopaedists, who use those prompting images to extend vocabulary, which are employed during teaching how to read. The problem with these prompting images is, that a 4 year old child cannot put into words what he sees in the picture, and if he can, he will not find any semantic relations between the 40 images, so as a result of regular repetition for a log time he will learn them as 'discreet entities'.

Our aim in creating a semantic network dictionary is that on the basis of a large body of empirical data we could show what other concepts can the most easily be associated in the STM of a 4—8 year old child to an input concept, i.e. how a semantic structure evolves in the STM. The weighted relations of semantic structures will make it predictable what words can the most easily and the most swiftly be associated with the limited vocabulary of children with dysphasia, i.e. what new theraphic opportunities are offered by the semantic network dictionary.

In my paper I will attempt to show how the dictionary is developed, what are the already achieved results and what further tasks are there to be solved.

2 The Semantic Structure

Memory research has proved [3] that those concepts can be acquired more easily to which we can associate some preliminary knowledge, than those ones to which we cannot associate anything. One stage of learning a new concept can be, that the new idea entering the SMT retrieves 1—6 such ideas (preliminary knowledge) from the LTM, which can some way be associated with the new one. (If the new concept does not retrieve anything from the LTM, it is either deleted, or a 'go-between' concept should be sought for, which can associate the new concept with a concept stored in the LTM, so this triple relation can trigger off the learning of a word.)

I call semantic structures the relations evolving in the LTM (either retrieved by exterior or interior stimulus), and with it I mean such a network of concepts, which is created in the LTM by retrieving 1—6 such concepts from the STM, which can be further associated with the concept of the prompting stimulus [2]. (The semantic structure consists of the relation of 2—7 concepts, because at least the relation of 2 concepts may prompt a third, and the upper limit [7 concepts] is necessary, because the capacity of the STM is unable to deal with more [10]. Our research has up to now proved that the STM capacity of 6-year-old children is approximately 4—5 units.

If we assess with empirical examination what semantic structure evolves in the STM of a healthy 5—8 year old child in response to a prompting stimulus, then on the basis of the processed empirical data it will turn out what units a semantic structure consists of, and taking it into consideration, on the basis of it we can plan more effective (faster and shorter) therapies for children with dysphasia, than before.

3 The Process of Making A Semantic Network Dictionary

3.1 The frequency dictionary of word usage of 6 year old children

In order to ascertain the already evolved semantic structures in case of the given population we had to compile a frequency dictionary, which relevantly shows what words the given population uses and/or what words it comes across most frequently [1].

During the compilation we processed the total vocabulary of all the 13 textbooks for 1^{st} grade students warranted by the Ministry of Education. Our preconception was, that the frequency dictionary compiled on the basis of the vocabulary of the textbooks will relevantly show what words occur the most often during the process of the acquisition of visual language in case of a 6-year-old child.

After processing the whole material contained in the textbooks, we deleted those grapheme sequences, which are not elements of our mother tongue, but they are unavoidable sound sequences during teaching how to read. Then we deleted the non-content words. Those words were considered to be content words, which carry all three sides of their meaning (lexical, grammatical and pragmatic).

Number of grapheme sequences	27.297
Number of content words	12.226
Number of words occurring 10 or more times	1.953

Table 1: Quantitative aspects of the dictionary.

Some of the quantitative aspects of the dictionary are shown in Table 1.

To create the semantic network dictionary we use the 200 most frequently occurring nouns. (We use nouns, because their acquisition is of primary importance as compared to the other words. We will process the verbs as well, but from another aspect.)

3.2 Compilation of data

Kindergarten nurses, teachers and high school students in 20 different settlements of the country currently do the compilation of data for the semantic network dictionary. The subjects of the compilation are third year kindergarten children, second and third year primary school children. We should like to collect data from 5,000 children by asking them what comes to their minds in connection with the 200 most frequently used nouns. We elaborated sheets on each of which 5 words occur, e.g. Father, mother, man, tree, word. The leader of the survey asks the question: 'What makes a tree a tree?' If the child has already acquired the visual language, he answers in writing, if not, he answers orally and the survey leaders takes down,

what he says. We do not use any quantitative limits, all tasks begin with the following command: 'Tell me .../Describe these concepts...'

The data compiled and the results processed so far show that children answer to a call word with 2.8 words or word sequences on the average, from which the following can be drawn: the semantic network dictionary will be competed on the basis of about 2,000,000 processed words. Naturally, these are not different words. On the basis of the processed material so far, we can say that about 350 concepts associate with one call word on the basis of material collected from 5,000 children. (The fluctuation is quite remarkable: the call word 'mother' retrieves 741 different associations, 'word' does only 178, so far these are the two extremities.) There will also be overlaps in the case of a call word associated with several concepts, that is why we cannot predict how many concepts will be included in the semantic network dictionary.

In the following I will describe five words, the processing of which on the given corpus has been finished: (the numbers in brackets show how many concepts are associated with the given word):

Mother:	she loves me	parent	dear	love	I love her
	(1588)	(1508)	(1263)	(1127)	(875)
Father:	parent	he works	love	he loves me	dear
	(1213)	(920)	(874)	(825)	(623)
Man:	being	life	he lives	head	clever
	(1813)	(1083)	(1046)	(934)	(930)
Tree:	plant	being	leaf	branch	air
	(2175)	(994)	(711)	(587	(514)
Word:	speech	letter	sentence	sound	he speaks
	(908)	(875)	(787)	(772)	(685)

3.3 Network Theories

In order to be able to compile our dictionary we studied the semantic network theories and their criticism [4,7,13], and the thesauri.

The problem with thesauri is that they do not provide any information on the relation of meanings of different lexemes. Furthermore, they combine the lexemes originating from different accents (regional, social, professional, etc.) without any remarks [5].

Still, the criticism of network theories and thesauri proved to me, why a network model resembling these theories may be the most appropriate conceptual framework to elaborate the most effectual therapies. Critics are right in saying that network models are to overpowering, that simply the evolution of the relation between the concepts is interesting, but they are incapable of showing the relationship between the concept and reality, which is the fundamental task of semantics.

However, from the viewpoint of the most effectual therapy that could be the most important point, that within the given population what relations between concepts already exist, and the large body of data is capable of showing this. It

can show on the basis of the frequency index what other concepts can the most easily and most effectively be linked with a given concept, and then what other concepts can be associated with these and so on.

Though no abstract semantic rule prevails in the evolution of semantic structures, which would be able to tell anything about the lexical meaning of a word, but a kind of subjective knowledge, still the total of empirically summed up subjective knowledges may provide some information on the words' organisation of meaning. This may not only present the free associations allocated to relevant words, but will unearth such relations between lexemes, which are created by each individual, i.e. it reflects the collocation.

Our semantic network dictionary will show these collocations, which is called by Landman intersubjective world-knowledge [9], so it will present not only the relations between concepts, but the new concepts generated by these relations, too.

4 Working of the software

When the whole word material is processed, computer-programmers will create a programme, which will present all the relations between the elements of the network with weighted bias. If we write, or say a word into a microphone, which is an element of the dictionary, then this lexeme will appear in the middle of the monitor, and the other five words with weighted bias, which is in the most frequent association with the word on the basis of the processed empirical material. If we click on, or say it in the microphone any of the five words forming a semantic structure with the original input word, then this will appear in the middle of the monitor as a call word together with the five most frequently associated concepts. As the software contains all the possible associations, you can go through the total vocabulary by evoking all the semantic structures in the middle of the screen. What we have said so far is shown schematically in Figure 2. taking the call word 'water' as an example.

The dictionary will be provided with a real-time speech-recognition system, which makes it possible for the children already capable of reading to take a virtual tour on the associated words of the network dictionary. This 'tour' will help enhance these semantic structures to develop and maintain in the memory of the child; i.e. it helps learning words.

We already possess a real-time speech-recognition system. Our research team recorded on computer 100 word inputs in eleven demonstration schools of the country, respectively from 500 subjects (6—7 year old children) on the basis of 1953 words (this is the number of words occurring 10 or more times) contained in the first grade textbooks. The recorded words were segmented and annotated. This material provides the basis for our software 'BeszédMester' [11], which can be employed for development in speech therapy and reading capability.

The basis for the speech-recognition module of our dictionary will be the nearly 250,000 segmented and annotated speech-sounds and the 50,000 recorded words (which includes the 200 most often used nouns serving as a base for the semantic

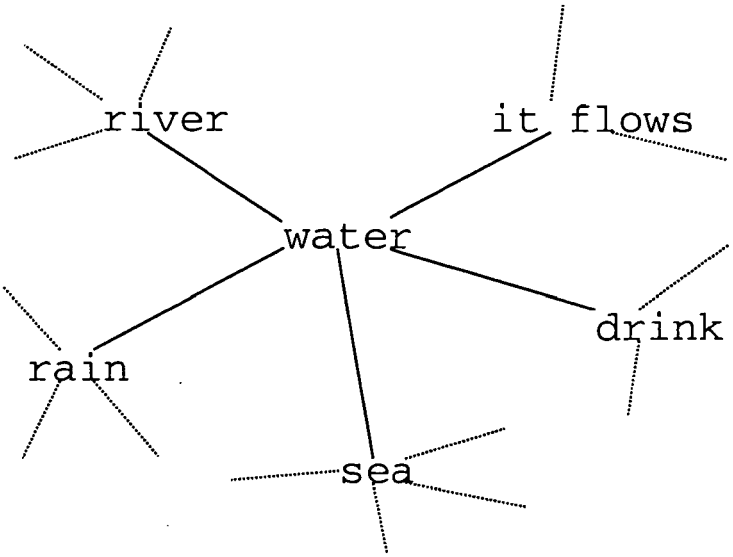


Figure 1: *A portion of the semantic network*

network dictionary).

We expect our semantic network dictionary to help therapist to plan development for children with expressive language disorder, as well as provide aid in expanding the vocabulary of healthy children.

5 Expected results of the research

The main purpose of the dictionary, as it was repeatedly mentioned, is to effectively aid the therapy of dysphasic children. In addition to the main purpose, several other aspects emerged during the project that may supply material to other researches. One such aspect is corpus linguistics since it would be impossible to process such an enormous database (nearly 3,000,000 words) in a short time without the advancement of informatics. It would be inconceivable to create all the empirically measured associations between the processed words. The complete survey and modeling of the associations may encourage further researches in child language, semantics, morphology and informatics as well as in pedagogy. The dictionary is innovative in both its form and functioning. There are already electronic dictionaries but this will be completely new in that it will allow unrestricted virtual walks through the whole network either manually or auditorily (by talking to the microphone) with immediate feedback about the relevant semantic structures. The Semantic Network Dictionary will seek its place in a cognitive theory not only because "association has its renaissance in cognitive psychology" [12] but also because a lot of cognitive processes are uninterruptible without postulating semantic

structures evolving from associations. Without aiming at completeness, only a few cognitive processes will be listed here whose semantic structures are built up from associations. Barkochba, an elaborate form of the Twenty Questions quiz game. is very popular in Hungary. When one is attempting to find out the thought of entity continuously narrowing a huge conceptual network, they are looking for some relevant concepts (4-7 in our assessment) to build the semantic structure from the associations that will designate the given entity. A similar situation is when one has to understand and/or learn a concept of which one has no information in their mental lexicon. In that case, one turns to "external memory devices," which are written, printed or electronic information storage media accumulated by mankind since the acquisition of visual language. The cognitive and evolutional aspects of external memory devices are discussed in detail by Donald [6]. External memory devices are used in a process in which a person's own knowledge of the world is being adjusted to the knowledge stored in external memory devices until the elements of the two knowledge structures build a semantic structure that makes it possible to define, understand or learn the given concept and put it in the LTM. External memory devices are scanned for data until a relevant concept in the person's memory is connected with an element of the external memory to create a semantic structure. The unit or part of the structure that can be connected to a unit or part of the external memory will be referred to as mediating semantic structure. The empirical studies of the role of the mediating semantic structure in cognition have just begun but one might already assume that all three semantic structure types will be found in every person's semantic network dictionary, i.e. there will be semantic structures that will be equivalent with the whole semantic structure, there will be mediating semantic structures and there will be semantic structures that will only contain the semantic structures of the external memory devices. We assume that the virtual walks in the dictionary will help in the process of learning by reducing the number of the semantic structures in the dictionary that are unknown to the individual.

6 Summary

Computer assisted study – whatever the object of study is – is an enormous motivating factor. Nowadays children learn earlier to use the computer, than to read or write.

Computer assisted study requires such didactics, programmes, which have not been part of institutional education and speech therapy yet. The search for new didactics, creation of programmes may provide a new conceptual framework for a scientific research. Computer assisted study modifies the framework of science(s) in order to apply it to its own research purposes. This is what we achieved when we re-evaluated the semantic network theories, as our aim was not the creation of a new semantic theory, but the development of plans capable of being used in the field education. Whether there exists a semantic structure, or can the examination of the association network be regarded as a scholarly undertaking is not our task

to decide. Practice provides us with convincing evidence about the existence and build-up of semantic structure. Supposition of its conscious build-up is useful for the methodology of speech therapy and vocabulary development.

That beyond the practical use of our suppositions there might be some scientific foundation can be proved by the fact, that Pléh speaks about the renaissance of association in cognitive psychology [12], and suggests that '... association and structure or logic are equally to be used in modelling higher processes.'

References

[1] Bácsi J. and Kerekes J. (2003): Az elsőosztályos olvasókönyvek szóanyagából készült gyakorisági szótár. "Van szó". [Frequency Dictionary Based on the First Grade Textbooks. 'The word be'.] Módszertani Közlemények. [Proceedings in Methodology.] 2003. II. 52–57.

[2] Bácsi J. (2003): A hálószemantika szerepe a megkésett beszédfejlődés terápiájában. [The Role of Network Semantics in the Therapy of Speech Development.] Alkalmazott Nyelvészeti Konferencia Füzetei. [Proceedings of Conference on Applied Linguistics.] 274–283.

[3] Baddeley A. (2001): Az emberi emlékezet. [Human Memory.] Budapest, Osiris Publishing House.

[4] Collins A. M. and Loftus G. R. (1975): A Spreading Activation Theory of Semantic Processing. Psychological Review, 82. 240–247.

[5] Crystal D. (1998): A nyelv enciklopédiája. [An Encyclopaedia of Language.] Budapest, Osiris Publishing House.

[6] Donald M. (2001): Az emberi gondolkodás eredete. Budapest, Osiris.

[7] Jonson-Laird P. N., Herrmann D. J. and Chaffin (1984): Only Connections: A Critique of Semantic Networks. Psychological Bulletin, 96, 292–315.

[8] Kerekes J. (2003): BeszédMester. [SpeechMaster.] Alkalmazott Nyelvészeti Konferencia Füzetei. [Proceedings of Conference on Applied Linguistics.] 44–49.

[9] Landman F. and Veltman F. (1982): Varieties of Formal Semantics. Proceedings of the 4. Amsterdam colloquium, Dordrecht, Holland.

[10] Miller, G. A. (1956): The Magical Number Seven, Plus or Minus Two: Same Limits on our Capacity for Processing Information. Psychological Review, 63, 81–97.

[11] Paczolay D., Tóth L., Kocsor A., and Kerekes J. (2002): Gépi tanulás alkalmazása egy fonológiai tudatosság-fejlesztő rendszerben. [The Use of Machine-Assisted Study in a System for Development of Phonology-Consciousness.] Alkalmazott Nyelvtudomány [Applied Linguistics] II(2), 55–67.

[12] Pléh Cs. (2003): Az asszociáció renaszánsza a kognitív pszichológiában. [The Renaissance of Association in Cognitive Psychology]
`http://sol.cc.u-szeged.hu/~pleh/magyar/cikkek/regi/4asszren.html`

[13] Quillian, M. R. (1969): The Teachable Language Comprehender: A Simulation Program and Theory of Language. Communication of the ACM, 12. 459–476.

Received May, 2004

Acta Cybernetica **16** (2004) 611–623.

Noun Phrase Recognition with Tree Patterns*

András Hócza†

Abstract

This paper offers a method for the noun phrase recognition of Hungarian natural language texts based on machine learning methods. The approach learns noun phrase tree patterns described by regular expressions from an annotated corpus. The tree patterns are completed with probability values using error statistics. The noun phrase recognition parser tries to find the best-fitting trees for a sentence using backtracking technique. The results are used in an information extraction toolchain.

1 Introduction

Noun phrase (NP) recognition is the process of determining whether sequences of words can be grouped together with nouns. NP recognition is an important part of the field of syntactic parsing but, to date, there is no suitable syntactic parser available for the Hungarian language.

Hungarian is an agglutinated language with a rich morphology and relatively free word order, whose properties add difficulties to the full analysis of the Hungarian language compared to Indo-European languages. These difficulties mean that the automatic NP recognition of Hungarian language is too complicated to solve using experts' rules only. An efficient solution for this problem might be the application of machine learning methods, but it requires a large number of training and test examples of annotated NPs. Since the Szeged Corpus[1] [4] became available, new methods have begun to be developed for syntactically parsing Hungarian sentences. The corpus contains texts from five different topic areas and is currently comprised of about 1.2 million word entries, 145 thousand different word forms, and an additional 225 thousand punctuation marks. Initially, corpus words were manually POS tagged and disambiguated by linguistic experts. Later, texts from the Szeged Corpus were parsed, where annotators marked noun phrase structures and clause structures. The extensive and accurate manual annotation of the texts, which required 124 person-months of manual work, is a good feature of the corpus.

*Presented at the 1st Conference on Hungarian Computational Linguistics, December 10–11, 2003, Szeged.

†University of Szeged, Department of Informatics, 6720 Szeged, Árpád tér 2. E-mail: hocza@inf.u-szeged.hu, Webpage: http://www.inf.u-szeged.hu

[1]Magyar Távirati Iroda, http://www.mti.hu

After the completion of the annotation work the Szeged Corpus was then used for training and testing machine learning algorithms to retrieve NP recognition rules. This paper introduces an application of the RGLearn [6] algorithm that was used to learn NP tree patterns described by regular expressions. The NP tree patterns are completed with probability values using error statistics. The NP parser uses this grammar to build up the best series of NP trees of a sentence by backtracking. The results look fairly promising after comparing them to related works. This method was developed as a part of a system which extracts information from short business news texts written in the Hungarian language.

This paper is organized as follows. Section 2 describes the difficulties of automatic NP recognition in the Hungarian language. In Section 3 there is a review of related works and a mention of efforts made by Hungarian researchers. Section 4 introduces a large annotated corpus that was used as source of training and test data. Section 5 then presents the method used for learning grammar from an annotated corpus and extending grammar with probability values using test statistics. Section 6 introduces our method of noun phrase recognition. Section 7 presents the test results. Lastly, conclusions and suggestions for future study are given in Section 8.

2 Difficulties of Hungarian Noun Phrase identification

Hungarian is customarily defined as an agglutinative, free word order language with a rich morphology. These properties make its full analysis difficult compared to Indo-European languages. Unambiguous marks for the automatic recognition of NP boundaries do not exist. The right bound of NPs (NP head) could be the nouns, but there is a possibility of replacement of NP heads with its modifiers. Determining the left bound of NPs is harder than the NP head, because it could be a determinant element. However, due to the possibility of a recursive insertion, it is not easy to decide which determinant and NP head belong together. Some of these difficulties can be illustrated in the following short sentences:

These difficulties mean that it is a hard problem to perform automatic NP recognition with experts' rules. In many cases the decision of annotators is based on semantic reasons rather than syntactic or structural ones. Another approach for automatic NP recognition is to use machine learning methods, but this requires a large number of training examples. In the past there was no large corpus for the Hungarian language containing annotated NPs.

3 Related works

Several authors published results of NP recognition parsers especially made for English. Generally the performance is measured with three scores. First, with a percentage of detected noun phrases that is correct (precision). Second, with a percentage of noun phrases in the data that is found by the classifier (recall). And

Free word order:

[Péter] olvas [egy könyvet]. (Peter is reading a book.)
Olvas [Péter] [egy könyvet]. (Peter is reading a book.)
[Egy könyvet] olvas [Péter]. (Peter is reading a book.)

Missing determiner:

[Péter] olvas [**egy** könyvet]. (Peter is reading a book.)
[Péter] [könyvet] olvas. (Peter is reading a book.)

Missing NP head:

[Péter] [a sárga könyvet] olvassa, [Mari] pedig [**a pirosat**].
(Peter is reading the yellow book and Mary is reading the red one.)
[a pirosat] = [a piros **könyvet**]

Figure 1: Examples of problems in Hungarian NP identification

third, with the $F_{\beta=1}$ rate which is equal to 2*precision*recall/(precision*recall). The latter rate has been used as the target for optimization.

Abney [1] proposed an approach which starts by finding correlated chunks of words. Ramshaw and Marcus [11] built a chunker by applying transformation-based learning ($F_{\beta=1}$=92.0). They applied their method to two segments of the Penn Treebank [8] and these are still being used as benchmark data sets. Argamon [3] uses memory-based sequence learning ($F_{\beta=1}$=91.6) for recognizing both NP chunks and VP chunks. Tjong Kim Sang and Veenstra [13] introduced cascaded chunking ($F_{\beta=1}$=92.37). The novel approaches attain good accuracies using a system combination. Tjong Kim Sang [14] utilized a combination of five classifiers ($F_{\beta=1}$=93.26).

So far, there is no good-quality NP parser published for Hungarian. The first report on the ongoing work of a Hungarian NP recognition parser [16] is based on the idea of Abney [2] using a cascaded regular grammar. The input to the grammar was a morpho-syntactically annotated text using a scaled-down version of the annotation scheme developed for the Hungarian National Corpus [15]. The grammar was developed by linguistic experts with the help of the CLaRK [12] corpus development system on a text of 928 sentences containing 23991 tokens. These samples were taken from a quality weekly economics journal concentrating on short business news items, which were suitable for an information extraction project. The rules of grammar contain context free patterns of NPs described by regular expressions. The patterns may refer to patterns of morpho-syntactic codes, words or previously recognized NPs. The rules were grouped together by linguistic experts into distinct stages which were applied cyclically by the CLaRK system using a bottom-up tree building technique. A 100-sentence text chunk containing 2537 tokens was produced by manual annotation to serve as the gold standard containing 488 NPs. The system parsing the test set found 611 NPs, of which 323 NPs were correct ($F_{\beta=1}$=58.78).

4 Training data

In order to perform well and learn from the various *Natural Language Processing* (NLP) tasks and achieve a sufficient standard of *Information Extraction* (IE), an adequately large corpus had to be collected that serves as the training database. During the setting-up of the various NLP projects a relatively large corpus of different types of texts was collected, called the Szeged Corpus [4]. For demonstration purposes in the above-mentioned NKFP project, the authors chose a collection of short business news items issued by the Hungarian News Agency[2]. The selected 6453 articles form part of the Szeged Corpus and relate to Hungarian or joint companies' financial and business life. The Szeged Corpus contains 1.2 million text words, 225 thousand punctuation marks, and comes in an XML format using the TEIXLite DTD (Document Type Definition). The first version of the corpus contains texts from five topic areas, roughly 200 thousand words each, meaning a text database of some 1 million words. The second version was extended with a sample of texts of business news totalling another 200 thousand words. The texts are divided into sections, paragraphs, sentences and word entries.

Initially, corpus words were morpho-syntactically analysed with the help of the HuMor[3] automatic preprocessor and then manually POS tagged by linguistic experts. The Hungarian version of the internationally acknowledged MSD (Morpho-Syntactic Description) scheme [5] was used for the encoding of the words. Due to the fact that the MSD encoding scheme is extremely detailed (one label can store morphological information on up to 17 positions), there is a large number of ambiguous cases, i.e. roughly every second word of the corpus is ambiguous. Disambiguation therefore required accurate and detailed work. It required 64 person-months of manual annotation. Currently all possible labels as well as the selected ones are stored in the corpus. About 1800 different MSD labels are used in the annotated corpus. The MSD label corresponds to the part-of-speech determined attribute, and specific characters in each position indicate the value for that attribute.

For example, the MSD label Nc-pa can be understood as

POS: Noun,
Type: common,
Gender: - (not applicable to Hungarian),
Number: plural,
Case: accusative.

In general, the NP building process of a sentence produces detailed NP trees much like Figure 2. These general NP trees must be simplified because, of course, simpler trees more readily support information extraction. This simplification was done by linguistic experts in the manual annotation phase. Hence the training corpus contains simplified NP trees. However the morpho-syntactic labels of corpus for POS tagging are more informal using the MSD encoding. The sentences of the corpus are stored like in Figure 3.

[2]MTI, Magyar Távirati Iroda (http://www.mti.hu), "Eco" service.

[3]The HuMor morpho-syntactic analyser is a product of MorphoLogic (Budapest) Ltd..

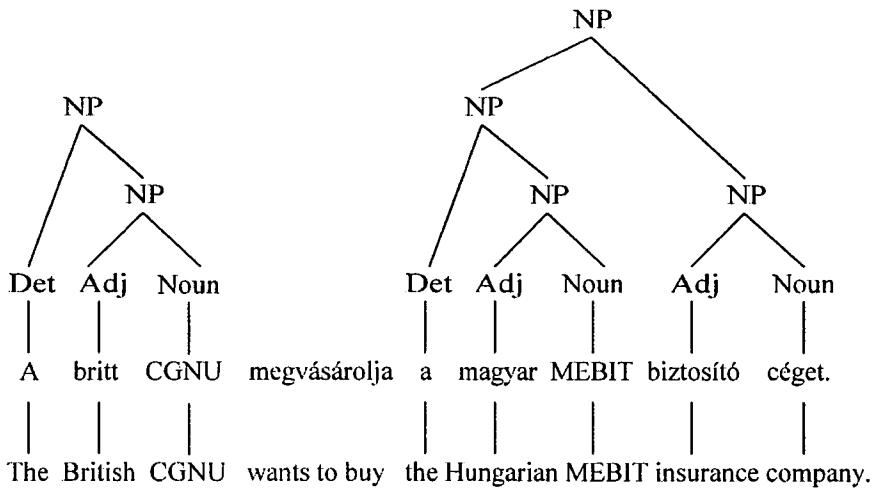


Figure 2: NP trees of a Hungarian sentence (with its English equivalent) from a short business news item.

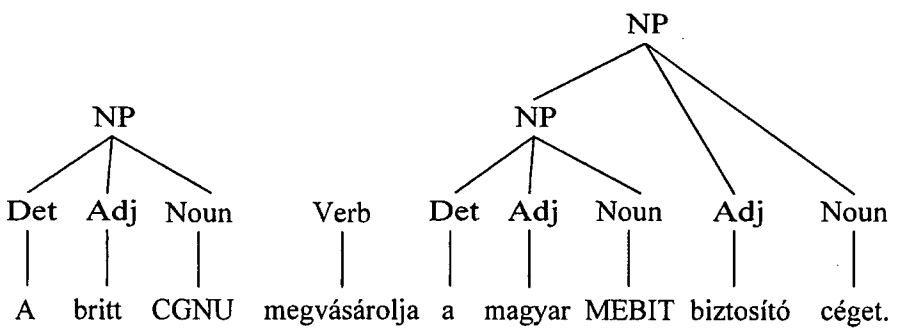


Figure 3: Simplified NP trees with MSD encoding from the sentence in Figure 2.

5 Learning tree patterns

In this section the learning task of noun phrases will be described which contains the preprocessing of training data, unification of tree patterns, searching for repetition of POS tag labels in tree patterns, handling of equivalent sub-trees and, finally, the complete unification algorithm. An application of RGLearn [6] is used as an NP tree learner. RGLearn is a rule generalization method based on a previous algorithm called RAPIER [9]. The RAPIER (Robust Automated Production of Information Extraction Rules) system learns rules by using a combined bottom-up and top-down learning algorithm. RAPIER is a modified version of the GOLEM [10] machine

[Tf Afp-sn Np-sn]
[[Tf Afp-sn Np-sn] Afp-sn Nc-pa]

Figure 4: Patterns from the sentence in Figure 3.

learning method. GOLEM uses the LGG (Least General Generalization) method for compression, i.e. it always compresses two examples, whereas RAPIER always generalizes two randomly selected rules.

5.1 Preprocessing of training data

The initial step for generating training data is to collect every complete NP tree from an annotated training corpus.

Complete NP tree means: the NP tree can contain other NPs but cannot be contained in another NP.

The collected training examples are complete context-free NP trees without words. For example, the sentence in Figure 3 has three NPs and two complete NP trees. The description of NP trees contains only lexical codes and NP begin and end marks ([,]), like the examples in Figure 4.

5.2 Unification of tree patterns

The collected NP tree patterns provide useful rules for NP recognition. With the help of these rules the NP parser is able to reconstruct the NP structures of training sentences. But, in order to perform the NP recognition of an unknown text to a fair accuracy, the collected NP tree patterns must be generalized. Generalization means the compacting of the rule set and expanding its coverage to include similar cases that do not occur in training examples. Various methods are used in the generalization learning phase. One of these methods is the most general unification of the rule set, which is the unification process of every possible similar rule one step at a time until similar rules exist in the rule set.

The meaning of similarity between rules is given by the following definition: rules R_i and R_j are similar if

- R_i and R_j contain the same number of tags,
- the first letters of the tags are the same in each rule position, which means NP start and end tags ([,]) and POS codes of words are the same,
- $\left(\sum_{t=1}^{T} Dif\left(R_{it}, R_{jt}\right)\right) \leq threshold$

where R_{it} is the t'th tag of rule R_i, T is the number of tags in the rules, Dif is the number of different letters of two tags, and finally *threshold* is a predefined value for the maximum difference. In general the *threshold* value is 1 or 2. The unification of similar rules means its replacement with a new, more general rule. Different parts in similar rules are changed to macro characters ('?', '*') in new rules, like the example in Figure 5. The symbol '?' in a pattern covers any letter.

Similar patterns:

[[Tf Afp-sn Nc-**sn**] Afp-sn Nc-sn]
[[Tf Afp-sn Nc-**pn**] Afp-sn Nc-sn]

Unified pattern:

[[Tf Afp-sn Nc-?n] Afp-sn Nc-sn]

Figure 5: The unification of similar patterns. The POS codes of Tf, Afp-sn and Nc-sn correspond to Det, Adj and Noun.

5.3 Repetition of POS tag labels in tree patterns

The next possible step in compacting the rule set is to reduce repetitions using the following generalization assumption for these cases: if some part of the pattern is repeated twice, it may be repeated infinite times in similar patterns of other examples taken from the training or test data. A simple repetition is a group of similar POS tags. The compacted pattern for this group is a label - the most general unification of similar POS tag labels. The regular operator '+' indicates that this tag can be repeated an indefinite number of times. Complex repetitions are when more than one similar POS tag label can be placed into the part of the tree pattern with free word order. To detect complex repetitions, examples of similar tree patterns are needed with a different POS tag label. In unified patterns the operator '|' (logical or) indicates that the set of POS tags can be used at that position of the tree pattern. Simple and complex repetitions can also be combined which are based on similar tree patterns as well. Examples of simple, complex and combined repetitions and their compacted versions are shown in Figure 6.

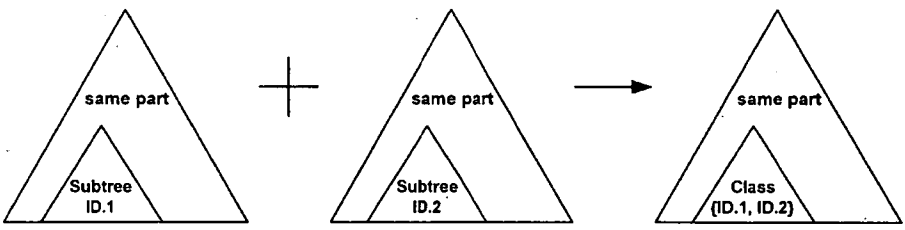
Kind	**Repetition in patterns**	**Unified patterns**
Simple	the greatest Hungarian insurance company [Tf **Afc-sn Afp-sn Afp-sn** Nc-sn]	[Tf **(Af?-sn)+** Nc-sn]
Complex	the great and famous company [Tf **Afp-sn Ccsw Afp-sn** Nc-sn] [Tf **Afc-sn Afp-sn Afp-sn** Nc-sn]	[Tf Af?-sn **(Afp-sn|Ccsw)** Afp-sn Nc-sn]
Combined	[Tf **(Af?-sn)+** Nc-sn] [Tf **Af?-sn (Afp-sn|Ccsw) Afp-sn** Nc-sn]	[Tf **(Af?-sn|Ccsw)+** Nc-sn]

Figure 6: The unification of repetition using similar patterns.

5.4 Equivalent sub-trees, sub-tree classes

Repetitions like those described previously do not contain NP bound marks ('[', ']') because preserving the NP tree structure is very important. NP bound marks are not elements of patterns because their function is the separation (and recognition) of trees and sub-trees. The usage of sub-trees for NP tree building requires that NP identification distinguish one NP from another. NP tree building without NP identification often gives rise to overgeneration, i.e. the generation of impossible tree structures for a given natural language. Hence it cannot be assumed that each NP is equivalent. But there are equivalent sub-trees when the context of two sub-trees is the same in their bigger trees (see Figure 7). Using the above notation the unification for sub-trees is given by the following process:

- Selecting rules that contain equivalent sub-trees, as in Figure 7.
- Collecting equivalent sub-trees to equivalence classes.
- The replacement of equivalent sub-trees with its equivalence class identification code. This means that bigger trees will be practically identical, so the unnecessary rules can be deleted from the rule set.



Example:

Similar patterns	Unified patterns
[[**Tf Np-sn**] Afp-sn Nc-pa] [[**Tf Afp-sn Np-sn**] Afp-sn Nc-pa]	[**CLASS.1** Afp-sn Nc-pa] CLASS.1: ID.1 - [Tf Np-sn] ID.2 - [Tf Afp-sn Np-sn]

Figure 7: The unification of similar patterns that contain different sub-trees and an example of unification.

5.5 The most general unification of the rule set

Bringing together the unification methods discussed in previous sections, the algorithm of the most general unification is quite simple: repeating unification methods one after another until something changes in the rule set. Expressed in formal terms, the procedure is the following:

repeat
unification of similar tree patterns
unification of repetitions
unification of equivalent sub-trees
until *(something changes in the rule set)*

5.6 Analysis of the grammar

The most general unification of the rule set allows the parser to be able to recognize every possible NP in an unknown text as well. But the cost of increasing generalization is decreasing accuracy. The accuracy for various rules is quite different: there are a lot of rules with very good (>95%)and very poor (<5%) accuracies. It seems obvious that bad rules ought to be dropped from the rule set (e.g. a rule with an accuracy of below 50% produces more errors than good recognition). But the strategy of an NP tagger is to find the best possible NP tree series for a sentence, so the rule with a poor accuracy (and low frequency) does not make so many errors because it is generally not chosen. For optimization the accuracy of NP recognition can be found experimentally. Which is the best threshold value for dropping rules because of its poor accuracy? According to the results shown in Figure 8, we should choose a rule accuracy threshold for dropping rules of about 5%-10%.

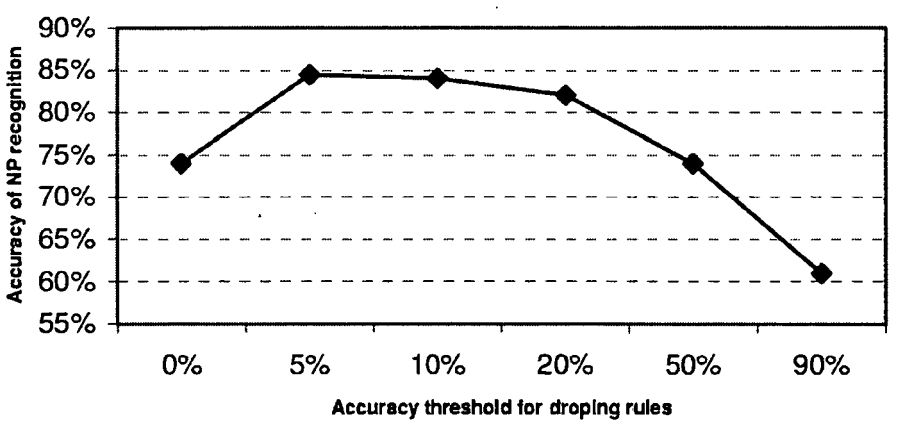


Figure 8: Where is the better threshold for dropping rules?

The main task of the analysis phase is to determine the probabilistic worth of rules and drop weak rules from the rule set using previous optimizations. The probabilistic value of a tree pattern is the accuracy normalized by the frequency. Summarizing the results of the learning and analysis phase is a context-free probabilistic grammar described with regular expressions for the recognition of NP trees.

6 Building of NP trees

The main task of the NP tagger is to find the best series of NP trees for input sen-tences. Input data contains disambiguated POS tag labels and it may come from the Szeged corpus at the testing phase of the method or it may be provided by a POS tagger tool [7] as a practical application of method. The following natural language processing (NLP) modules are used to determine linguistic features for NP recognition, as shown in Figure 9. They consist of tokenization, sentence segmentation, morpho-syntactic analysis and part-of-speech (POS) tagging.

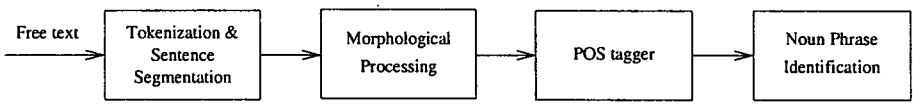


Figure 9: The system architecture of natural language processing pipeline

The NP processing is performed sentence by sentence, but sentences can be divided into smaller parts between verbs and the border of clauses. The usage of this division makes NP processing faster. The initial phase of tagging a sentence is to label all possible trees in every word position of a sentence. Naturally there is some overlap-ping among the labeled NP trees and a word position can be the beginning of many NP trees. The preferred attributes for evaluating NP trees are the probability, phrase length (number of word positions in NP) and depth of the NP tree. There is a back-tracking algorithm that searches for the best (or best of n) NP tree series. After finding the best series of NP tree patterns, the actual NP tree structure of a given sentence can be reconstructed (built up) from tree patterns. The NP recognition algorithm of a sentence involves the following:

BEST = nil
Labeling of each word position with possible matching tree patterns
repeat
    CURRENT = next combination of tree patterns
    if *(probability of CURRENT > probability of BEST)*
        then *BEST = CURRENT*
until *(not computed each possible combination of tree patterns)*

Text category	Precision	Recall	$F_{\beta=1}$
Corpus version 1.0	75.72%	81.69%	78.59%
Business news extension	79.86%	86.63%	83.11%
Business news extension	79.86%	86.63%	83.11%

Figure 10: Test results on the two corpus domains.

7 Results

The method presented for learning and recognizing NP trees was applied on two cor-pus domains: the first version of the corpus containing texts from five different topic areas and the second domain from the business news extension. The sentences of domains were randomly divided into train (90%) and test (10%) sets. The preprocess-ing phase collected from the first domain had 297,077 complete NP trees and the unification learner produced 7,476 NP tree patterns. The second domain contains 51,112 complete NP trees and 1,856 acquired NP tree patterns. The training phase associated probability values with the patterns. The evaluating of the grammar was performed on 10% of the test set. A summary of test results is shown in Figure 10.

So far, the results seem encouraging. Based on experiments, this NP tagger as an element of a natural language processing pipeline provided enough information for information extraction. There are some differences in the results of the two corpus domains because of their various characteristics. The business news part contains relatively homogeneous texts with often repeated idioms. The first domain is more heterogeneous in its five topics. After analyzing the reasons for errors it was found that, in many cases, extra semantic information was needed to select the proper NP tree structure. Overall, the recognizing of large (long and deep) NP trees seems to be quite a hard problem.

8 Summary and future work

In the current paper, the author presented a general learning method for NP recogni-tion that has been applied to learning and recognizing NP trees. The NP tree learning method was applied to a previous machine learning method RGLearn. The NP recog-nizing method, as an element of a natural language processing pipeline, provides suf-ficiently rich information to support information extraction.

The NP tree learner method id based on rule generalization that includes unifica-tion algorithms. The generated grammar is a set of NP tree pattern that is described with regular expressions. After the unification phase probabilistic values are added to the grammar using error statistic analysis of the training examples. The NP tree

tagger uses pattern matching and backtrack algorithm to search for the best-fitting NP tree sets. The NP tree structure of given sentence is reconstructed from tree patterns.

In the future, running parallel to the development of the Szeged corpus, the author intends to develop learning and recognizing method for constructing the syntax tree of sentences. The usage of ontological information that can be the extension of a mor-pho-syntactic description is also planned. The system will be primarily applied to the business news domain. In the 6th Framework Programme, an international research consortium is planning to develop a multilingual IE system to be applied to above-mentioned two domains.

References

[1] Abney S. (1991) Parsing by chunks, in Principle-Based Parsing. Kluwer Academic Publishers.

[2] Abney S. (1996) Partial Parsing via Finite-State Cascades, in Proceedings of ESSLLI'96 Robust Parsing Workshop, pp. 1-8.

[3] Argamon, S., Dagan, I., and Krymolowski, Y. (1998) A memory-based approach to learning shallow natural language patterns, in Proceedings of 36th Annual Meeting of the Associa-tion for Computational Linguistics (ACL), Montreal, pp. 67-73.

[4] Alexin, Z., Csirik, J., Gyimóthy, T., Bibok, K., Hatvani, Cs., Prószéky, G., Tihanyi, L. (2003) Manually Annotated Hungarian Corpus, in Proceedings of the Research Note Sessions of the 10th Conference of the European Chapter of the Association for Computational Linguis-tics EACL03, Budapest, Hungary, pp. 53-56.

[5] Erjavec, T. and Monachini, M., ed. (1997) Specification and Notation for Lexicon Encoding, Copernicus project 106 "MULTEXT-EAST", Work Package WP1 - Task 1.1 Deliverable D1.1F.

[6] Hócza, A., Alexin, Z., Csendes, D., Csirik, J., Gyimóthy, T. (2003) Application of ILP methods in different natural language processing phases for information extraction from Hungarian texts in Proceedings of the Kalmár Workshop on Logic and Computer Science, Szeged, Hungary, 1-2 October, pp. 107-116

[7] Kuba, A., Bakota, T., Hócza, A., Oravecz, Cs. (2003) Comparing different POS-tagging tech-niques for Hungarian in Proceedings of the MSZNY 2003, Szeged, Hungary, pp. 16-23

[8] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993) Building a large annotated corpus of English: the Penn Treebank, Association for Computational Linguistics.

[9] Muggleton, S. and Feng, C. (1992) Efficient Induction of Logic Programs, in Inductive Logic Programming (ed.: S. Muggleton), Academic Press, New York, pp. 281-297.

[10] Plotkin, G.D (1970) A note on inductive generalization, Machine Intelligence (eds: B. Meltzer and D. Michie), Vol 5.

[11] Ramshaw, L. A., and Marcus, M. P. (1995) Text Chunking Using Transformational-Based Learning, in Proceedings of the Third ACL Workshop on Very Large Corpora, Association for Computational Linguistics.

[12] Simov K. (2001) CLaRK - an XML-based System for Corpora Development, in Proceedings of the Corpus Linguistics 2001 Conference, Lancaster, pp. 553-560.

[13] Tjong Kim Sang, E. F., and Veenstra, J. (1999) Representing text chunks, in Proceedings of EACL '99, Association for Computational Linguistics.

[14] Tjong Kim Sang, E. F. (2000) Noun Phrase Recognition by System Combination, in Proceed-ings of the first conference on North American chapter of the Association for Computational Linguistics, Seattle, pp. 50-55.

[15] Váradi, T. (2002) The Hungarian National Corpus, in Proceedings of the Second International Conference on Language Resources and Evaluation LREC2002, Las Palmas de Gran Ca-naria, pp. 385-389.

[16] Váradi T. (2003) Shallow Parsing of Hungarian Business News, in Proceedings of the Corpus Linguistics 2003 Conference, Lancaster, pp. 845-851.

Received May, 2004

Acta Cybernetica 16 (2004) 625–642.

Extraction of Affective Components from Texts and Their Use in Natural Language Dialogue Systems*

Gábor Tatai† and László Laufer‡

Abstract

We are carrying out a research in the field of Human Computer Interaction and developing a natural language dialogue system in Hungarian. In the beginning chapters we briefly describe the architecture of our dialogue system, BotCom with examples of its semantic processing capabilities. We give examples of how the system is handling the topics of the discussion, how the dialogue history is being used in order to enhance the reply generation. In the subsequent parts we give an overview of the emotional state detecting, processing and generating module, called GALA, which is founded on the grounds of Robert Plutchik's emotional model. We show how BotCom is utilizing the detected emotional loads of the user's messages, therefore enabling the chatterbot to give relevant answers both semantically and affectively. In the final chapter we explain how the database of GALA was filled up with expressions assigned to their emotional loads. We also describe a graphical user interface (GUI) being designed to model the changing emotional loads in dialogues, songs and poems, and how it can be used for the emotional labeling of the phrases.

Keywords: Affective Computing, Emotional Modeling, Dialogue System, ECA, Chatterbot

1 Introduction

Many embodied conversational agents (ECAs) are targeting the Internet. However, systems that are bound to this global network not only benefit from several advantages of the huge amount of accessible information provided by this medium, but inherit its common problems as well. Among those are the difficulties of relevant search, complexity of available information, unstructuredness, lack of navigation support [3], bandwidth limitations and so forth. Another difficulty in order to

*Presented at the 1st Conference on Hungarian Computational Linguistics, December 10–11, 2003, Szeged.

†Department of Computer Science, University College London, Gower Street, WC1E 6BT, London, UK, e-mail: g.tatai@cs.ucl.ac.uk

‡Department of Psychology, ELTE University of Sciences, 1064 Izabella utca 46, Budapest, Hungary, e-mail: laufer@erg.bme.hu

achieve better believability of the agent is to make it capable of detecting the user's emotions and giving emotionally adequate responses [2].

Although many researchers recommended to apply emotions in artificial intelligence systems, not only in conversational systems, but generally in cognitive architectures and decision making systems [11, 6], it is still rare to find applications utilizing this type of cognition in problem solving. Also in modern chatterbot systems the designers do not really pay ample attention to emotional modeling, or the solution they use is far from being sophisticated.

For a human being a communication partner without even the simplest emotions can be very boring. Adequate detection and communication of emotions are essential in every day interactions [4], and human-computer interactions as well [17]. Not mentioning tutoring and general educational systems which are one of the primary potential application areas of the chatterbots. Probably most of us have had unpleasant experiences with poker-faced teachers at oral examinations, when there was no apparent sign of understanding, approval or disapproval. Therefore it is almost a mandatory requirement to equip an ECA with a reasonably refined emotional model. Such a communication partner can be more alive, convincing and entertaining as noted already in earlier experiments [17].

Thus we aim to develop a more sophisticated emotional module, enabling the treatment of the continuously changing emotional status during a conversation

There is no space to discuss all the features and interesting implementation experiences with our ECA system in this paper. Therefore we are only focusing on our system architecture with a special emphasis on the emotional module. We think others might find interesting and get some thought-provoking ideas in our theoretical approach and its practical implementation.

2 Architecture and Technical Considerations

Our ECA system called BotCom is primarily a client-server architecture with a chatterbot development module, BotMaker (Fig. 1.).

The ECA is embedded in web pages and displayed in a web browser. The client visualizes the messages of the chatbot, the animations representing its emotions and the gags, as well as forwarding users' messages to the server. Portal engine integration enables access to the indexed database of keywords and full articles of the site, providing extendibility to the knowledge base of the chatterbot.

On the server side a Java servlet collects the messages and transmits them to a multiplexer which queues or distributes them among the servers for parallel operation. Messages are processed separately, in each logged-in user's context, which are responsible for the communication and stores user preferences and personal details. The dialogue system matches them against the patterns stored in the knowledge base, and the answer, which consists of the status obtained from the GALA emotional model (see below) and the browser controlling commands (if any), are sent back to the client.

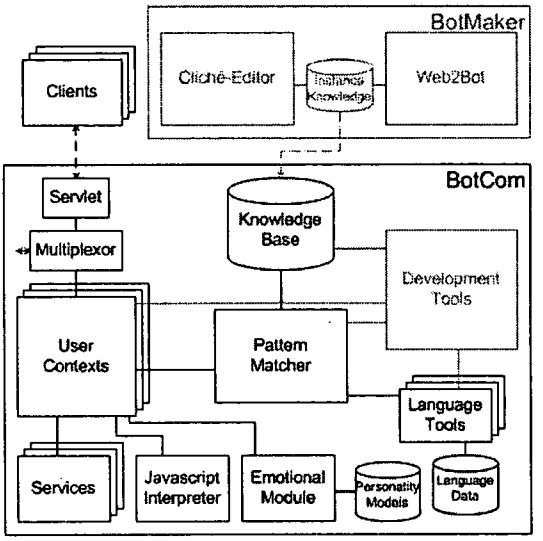


Figure 1: The main components of BotCom and the BotMaker service and development tools

Development tools for uploading and managing the knowledge base of the ECA and adding functionalities include the BotMaker and the Cliché Editor.

3 Message Processing

An ECA have to perform complex procedures, in order to give an adequate answer to the user message. The adequacy of the emotional load of the answer is a separate issue, we are discussing it in the description of the emotional module. On Fig. 2. we tried to give an overview of the response generation procedure.

As it is visible on the flowchart, the core of the system is a pattern matcher working on a knowledgebase of dialogue segments. One of the key issues when creating a communicational agent is to set up a structure and to establish its knowledgebase. On Fig. 3. we are showing the knowledgebase editor interface of BotCom, Botdev. The dialogues segments are structured into topics and subtopics, these are important to determine the semantic difference between to sentences. It was also important to give alternatives to each sentence entered, the meanings of the alternatives are the same, but they can differ in their emotional load. The interface enables its user to write scripts in a special script language to define the circumstances of giving the specific answer, it also gives an opportunity for testing the chatterbot with using the entry.

The incoming message is forwarded to the pattern matcher and also submitted to 3 procedures for further analysis (see Fig. 2.). Syntactic processing (on the right)

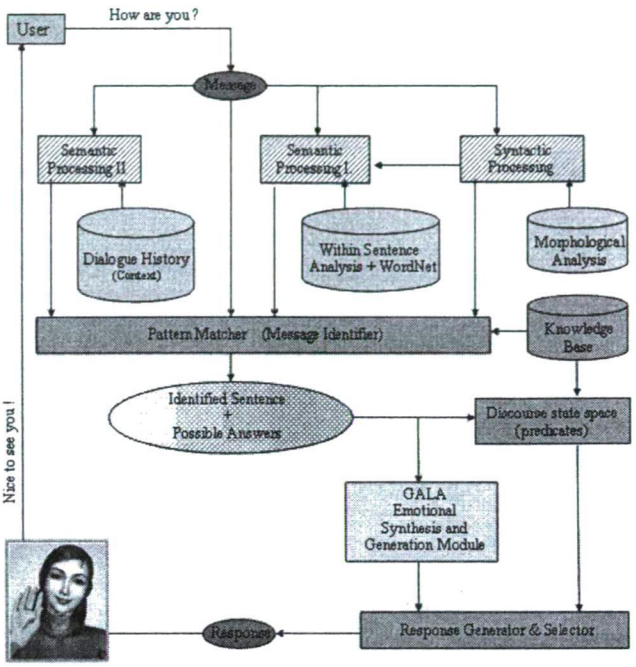


Figure 2: The message processing workflow of our chatbot system BotCom

makes sure, that the spelling mistakes are corrected, also other common mistypes (partly due to the difference between English and Hungarian keyboards) are taken into consideration. This sub-module is using morphological analysis tools available for Hungarian language. The output is forwarded to the first semantic processing sub-module (beside the pattern matcher).

The first semantic processing sub-module (see Fig. 2. upper-middle part) is responsible for semantic processing of words and phrases within the sentence. In order to carry out this task, it utilizes semantic-nets: OSZK's Thesaurus [13], and other free thesaurus dictionaries and WordNet databases. This way if the pattern matcher did not identify the incoming message in the knowledgebase, we are able to provide further suggestions with identical meaning to be matched among the existing dialogue segments.

The second type of semantic processing sub-module (see Fig. 2. upper-left part) is to identify the role of the sentence in the dialogue. We expect the incoming message to have certain attributes in a certain phase of a discussion. We utilize the dialogue history, and conclude which part of the knowledgebase can be excluded and which part should be considered with a higher weight in the pattern matching algorithm.

In the second phase of the response generation, after the pattern matcher identi-

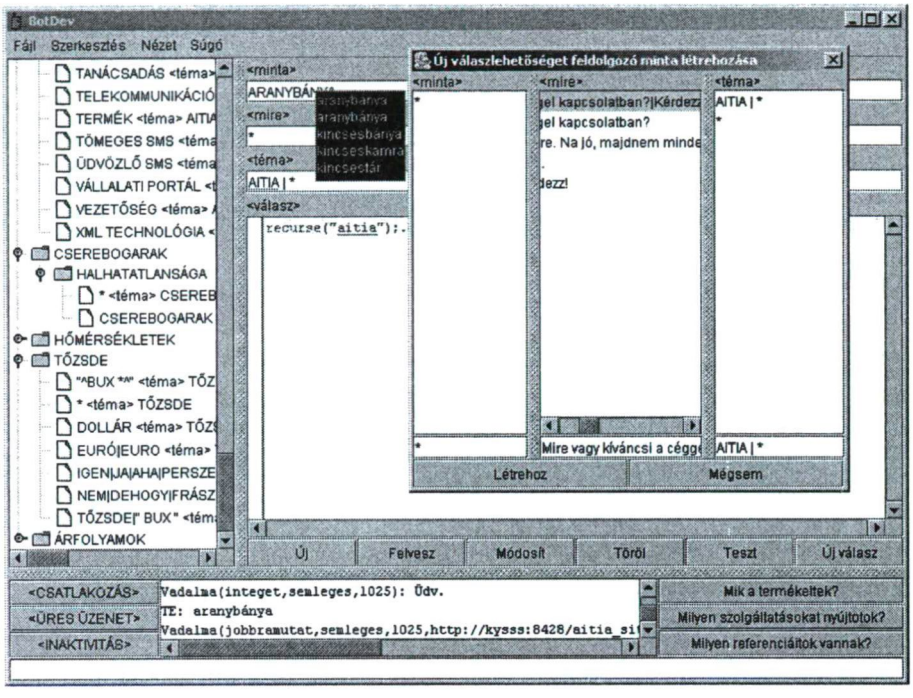


Figure 3: The knowledbase editor of BotCom

fied a sentence in the knowledge base, expectedly with identical semantical content to the incoming message, the pattern matcher gives the possible responses as well. The dialogue as a whole has a position in a state space. This position is determined mainly by different predicates describing the user (his habits, properties, other attributes assumed from this and the previous discussions) and the chatterbot (likes, dislikes, opinions stated previously) and other circumstances of the conversation. These are represented in the discourse space state, and influencing which possible answer is to be chosen from the different possibilities offered by the pattern matcher.

The Gala emotional module, described below, also processes the incoming messages and possible answers and chooses the best answer, which is emotionally adequate for the chatterbot to reply. Gala keeps track of the emotional aspect of the discussion. With its multi-layered architecture it is possible to model the emotional influences of the consequent user messages and to generate emotional moods for the chatbot, and also to create different personalities on the basis of which the ECA system gives different responses in similar situations.

On the basis of these information the response generator select an answer from the possibilities, and the chatterbot system displays it, together with the animation assigned to the message in its user interface.

4 Affective Computing

In this paragraph we would like to describe the emotional module of our system, but first say a few words about it's theoretical basis within artificial intelligence research.

For about a decade, the research area of Affective Computing has been flourishing. This field, as Rosalind Picard, a leading researcher in affective computing defined, deals with problems where "computing is related to, arises from or deliberately influences emotions" [16].

When researching into emotional computing one should certainly determine what is considered as an emotion. Since various recent advances in psychology and neurology are making cognitive scientists and psychologists to rethink the origins and behaviors of various emotions, there is presently no universally accepted emotional model.

On the other hand, there are some theories and foundations which we accept as basis. The early work on emotions by Darwin [5] and others (e.g. [10, 20, 19, 7]) who distinguished discrete categories, so-called basic emotions have resisted the test of time at least for their main categories and the description of these emotions. Others, however, have emphasized the continuous dimension of emotions [18], which is likely to be closer to the real operation of emotions. Nevertheless, it seems that the reflections on the emotions which identify them within the self, thus making the self aware of them, introduce a discrete element into the anyway continuous neurobiological system. The discreteness is even more noticeable when the emotion is translated to language and expressed so [12].

The most widely accepted set of basic emotions seems to be the eight basic emotions described by Tomkins (fear, anger, anguish, joy, disgust, surprise, interest and shame). This set is a reasonable extension of Darwin's first identification of the basic set (joy, anger, fear) [5], and, more or less, others' categories also correspond to it.

Beyond these, there is a well-known and widely accepted classification of emotions. This is the OCC Cognitive Model [15, 21] which provides not only a definition of them, but also a hierarchy based on the target of the emotion (self, other) and other meta-categories such as consequences of events, aspects and actions. This is a strict and complex cognitive model of the key emotions, however, in [14] Ortony himself admitted that the OCC model is far too complicated for the development of emotional characters and proposed a simplified model. In order to use this approach, one does not only need to model the emotions themselves, but it is necessary to adopt the entire cognitive model including all its processes and structures. In that case, we would be able to manage the emotions appropriately, however this would be very cumbersome. Even a simple model implies a complex architecture, as is clearly visible from reports on similar projects (Integrating Models of Personality and Emotions into Lifelike Characters) at DFKI [1].

5 The GALA Hierarchical Emotion Processing Model.

5.1 The Plutchik Model

Our goal was to select a sophisticated emotional model complex enough to provide ample workspace, but not bound to a net of cognitive processes. We have found that the model suggested by the psychologist Robert Plutchik is an appropriate starting point for further description and generation of synthetic emotions [22]. He defined eight basic emotions that are just a little different from those of Tomkins. In addition he suggested another dimension, intensity (with 3 levels). Therefore we get 24 emotional states. A novelty of his proposal is that the emotional space is mapped to an upturned 3D cone (see Fig. 4), where the positioning of a particular emotion reflects psychological distances and intensity differences between states. The origo is in the apex of the cone expressing neutrality ("no emotion"). We have selected this model for implementation and incorporated it as a part of our more complex, layered architecture. This geometrical approach allows easy manipulation of the emotions.

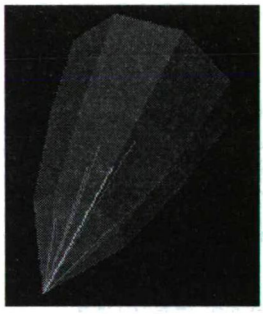


Figure 4: The 3D cone representation of emotions

5.2 The Architecture of GALA

In order to create a usable synthetic emotion model for a chatterbot that communicates mainly via text messages, it is essential to design an appropriate mapping scheme between the emotions and the expressions that the chatbot sends or receives (Fig. 5).

5.3 Layer One – Message Act Processing

The connection between the text messages, better said, message acts and emotions is thoroughly investigated by M. A. Gilbert in his work [9] and later in [8]. This work was particularly useful because it has pointed out that there is a reasonable

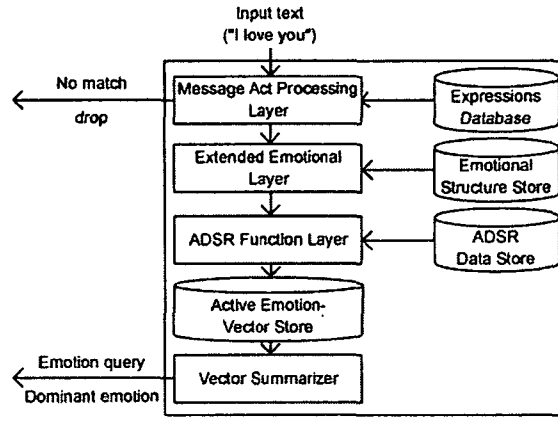


Figure 5: The architecture of the GALA engine and the processing of a message act

difference between the well known concept of a speech act and a message act that the meaning of the latter also carries emotional load. According to Gilbert's definition, a message act is similar to the speech acts; "A message act, being analogous to an utterance act involves an expression of emotion that is identifiable to the recipient or observer." (for example: threat, fear etc.). The speech act is used when analyzing sentences, query semantic information and emphasizes the verbal aspect of the communication. In contrast the message act focuses on various forms of the communication and used to express dialogues and emotional aspects in addition to the semantic, meaning level.

When processing the input expressions, initially they are passed to the GALA processing module. In the first step the conversational elements (words, expressions, sentences) in the dialogue are assigned to the emotional message acts, which are processed at the uppermost layer. If GALA finds a match of the input expression it passes to the second layer, otherwise the element is dropped and does not alter the emotions of the chatbot. The set of recognizable elements can be extended and the system can be trained.

5.4 Layer Two – Extended Emotional Layer

A message act almost never expresses a pure basic emotion. Therefore we have defined a second layer that allows the construction of mixed emotions that are more closely related to the message acts. In this layer one or more basic emotion components can be assigned to the individual message acts. These components can be weighted, so a message act could be represented by a mixture of e.g. 20 % sadness, 40 % anger, 40 % fear. There is a predefined database, the "Emotional Structure Store" (ESS) that allows the storage of these complex emotional structures and

their assignment to the appropriate message acts. The basic emotion components are represented by vectors and thus the resulting mixed emotion is also stored and manipulated in a vector. On the 3D surface suggested by Plutchik, the origin of these component vectors is the apex of the cone, and they point towards the center of the particular surface segments that are assigned to that basic emotion. In general we did not introduce a limitation on the number of basic emotions that can build up the resultant emotion which is associated with the message act. However, according to our experiments and anticipated real life requirements, it seems that there is a practical upper limit. It appears that most of the expressions can be realistically covered by a mixture of maximum 4 of the 24 predefined emotions in a given situation.

Summarizing: the function of the second layer is to break up the message acts into basic emotions taken from the Emotion Structure Store and forward them to the third layer.

5.5 Layer Three – ADSR Refinement Function

It is well known that emotions are time dependent, so we had to manage that the amplitude of their synthetic pairs change over time as well, since it is not handled by the original model. In addition, the characteristics of this change vary by emotion. As it was not our goal to replicate the unclear nature of the chemical and neurobiological change of emotions in the brain, we chose a new controllable approach, hoping to achieve a realistic result.

Therefore, in the third layer we used a well known function, called ADSR to describe the change of emotions over time. Signal processing and musical applications widely apply the method of dividing a signal to four main parts. Thus a cover curve of a signal can be defined by the Attack, Decay, Sustain and Release values (see Fig. 6.).

In our model all basic emotions can have their own ADSR functions, which ensure unique time characteristics for each of them. Moreover, the intensity of each emotion can be tuned in order to provide opportunity of the relative scaling of the emotions. So, in a particular situation the amplitude (intensity) of joy can be stronger than the intensity of the rest of the emotions still maintaining the same ADSR function and thus its behavior over time.

5.6 Computing of the Dominant Emotion

The ultimate goal of the process is to determine the dominant emotion since we have a strong constraint – we can play only one animation at a time. The third layer, after the decomposition of the message acts into emotional structures and further into basic emotions, selects the ADSR and intensity values corresponding to time t ("now") for each basic participating emotion. These emotion vectors are kept in the Active Emotion-Vector Store (AEVS). The actual resulting emotion vector is calculated by summing and normalizing these active vectors. This final vector will contain the effect of the actual message act (through its decomposed

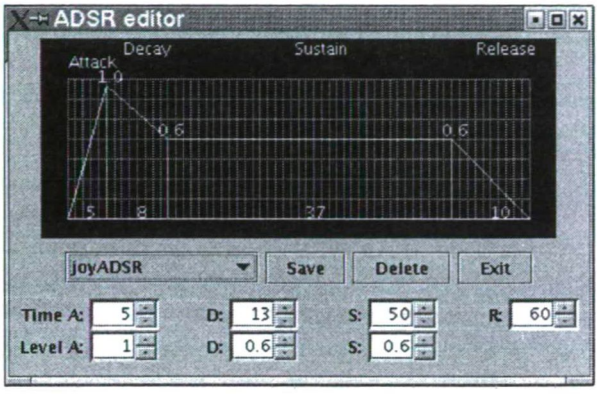


Figure 6: The ADSR function editor (It is available for each emotion or emotion structure).

participating emotions) and the fading effect of previous message acts which are already in the Active Emotion-Vector Store.

The processing of the normalized sum of the vectors produced by the three layers can be expressed in the following closed formula. Please note, that we are not calculating by merely using this formula, since it is a pipeline process and partial results must be calculated previously. The formula is provided for better understanding only.

$$\underline{r}(t) = \sqrt{\frac{\sum_{k=1}^{n} \underline{e}_k^2(t)}{n}} = \sqrt{\frac{\sum_{k=1}^{n} (i_k \underline{E}_k f_k(t - t_{0k}))^2}{n}}$$

$\underline{r}.$: resulting emotional state vector
t : actual time
n : number of active emotion vectors (where "active" indicates a non-zero vector)
$\underline{e}_k$: the k^{th} active emotion vector
i_k : the maximal intensity of $\underline{e}_k$
$\underline{E}_k$: unit vector of $\underline{e}_k$
f_k : ADSR function belonging to $\underline{e}_k$
t_{0k} : the activation time of the k^{th} emotion

The result vector can be drawn and it will point to some location within the cone. The dominant emotion for that t moment will be that basic emotion (out of 24) which is the closest to the endpoint of the vector sum.

As time goes, the system dynamically recalculates the emotions by repeating this decomposition process from layer one to three. Therefore a sequence and pattern will be displayed as emotions that can be traced on the emotion editor (see Fig. 4 later) appear and fade.

One may notice that our original standpoint was to create a fairly straightfor-

ward emotional model, however, it may seem to become moderately complicated. In fact, most of the components are quite simple and reused such as the emotion handling and the ADSR functions. The only thing that happened is that we have used them in multiple layers enabling complexity with simple, reused and overrideable components. This approach is similar to the theoretical and experimental results of psychology, where complex memory and emotional models arise from using very simple elements.

6 Administration Components of GALA

For administrating the GALA emotion module we created an application with graphical interface. The interface can load continuous texts, and visualize the emotional labels that are already stored in the emotional expression store assigned to the expressions in the content. This way through the administrator panel we can read out the text, and follow the changes of emotional states in time.

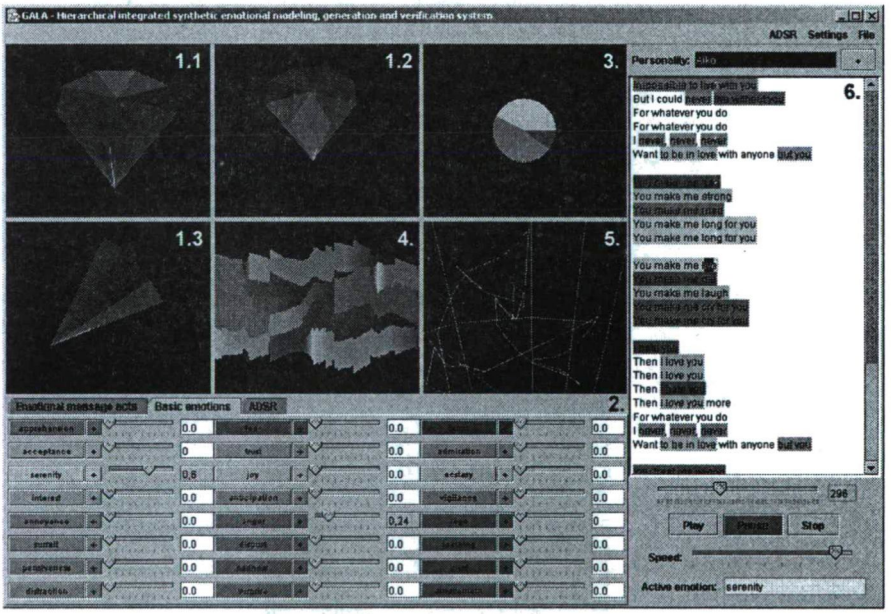


Figure 7: The emotion editor for administrating and testing the GALA model.

6.1 Tracing and Visualization Components of GALA

This interface allows the carrying-out of reproducible experiments on the emotional model, while the various visualization tools allow us to trace the process of emotion generation. In the following paragraphs we briefly describe its components.

6.2 Camera views

The three rotateable and zoomable camera views (see Fig.7. panel 1.1, 1.2, 1.3) display the cone with the vectors (active emotion vectors and the resulting emotion vectors) inside. These vectors are representing the different emotional influences caused by the previous passages, they are constantly changing, and fading away as a result of the ADSR function manipulates their length. The resulting emotional vector is also being displayed with the thicker white vector visible on all three panels.

6.3 Basic emotions monitor and Text panel

The basic emotion monitor shows the intensities of each basic emotion at a given moment (see Fig.7. panel 2.). It provides a great help to refine the basic emotions assigned to each phrase in the text. The cursor on the Text panel (see Fig.7. panel 6.) shows us which part of the text is being currently read. On the Basic emotion monitor it is possible to readjust the intensity of the 24 basic emotions participating in the composition of the complex emotional load of the current phrase. At the button part of the Text panel (see Fig.7. panel 6.) the user can start, stop and choose a position in the reading or adjust the speed for better following the changes in the resulting emotion vector. The dominant emotion (or active emotion) is calculated from the projection of the resulting emotion vector to the surface segments of the cone and being displayed at the button left corner of the panel. This emotional state is the output of the GALA emotional subsystem.

6.4 Active emotion component diagram

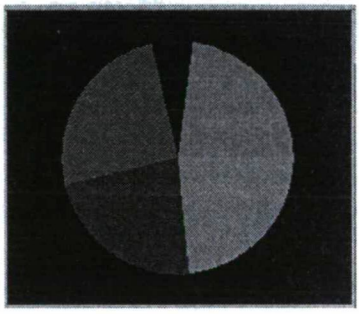


Figure 8: Active emotion component diagram.

Besides the camera views, there is another possible way to track and display active emotions (see Fig. 8). The sectors of the circle symbolize the active emotion components at any given time and their colors correspond to that of the appropriate emotion. The areas of the sectors are directly proportional to the percentage

shares of the basic emotions from the current resulting emotion vector, the present composite-emotional state. The length of this vector, the intensity of the composite emotion is represented by the radius of the circle.

6.5 Emotion-history panel

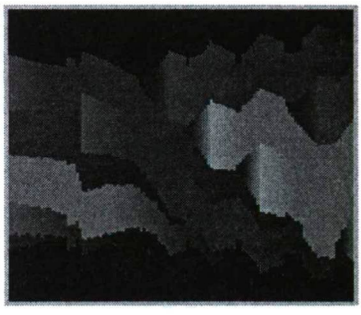


Figure 9: Emotion-history panel.

This panel has an important role in displaying the emotions synthesized throughout the whole conversation. When "reading" the text, in this case a poem, the system generates a colorful pattern that gives us a picture of the poem and enables us to get an overview of the emotional characteristics of the analyzed poem (see Fig. 9).

The X axis of the pattern stands for the time dimension: a vertical slice shows the active emotions (represented by the appropriate colors) at a given time, proportionate to their actual intensity. The height of a vertical slice represents the length of the resulting emotion vector.

As time goes by, new slices ("emotion ensembles") are added to the picture from right, resulting in the continuous condensation of the existing piece.

6.6 Emotion tracing panel

This panel allows us to follow and depict the route of the resulting emotion (the sequence of emotional change) of the conversation, in the function of time. For better visualization, we used only the wire-frame view of the Plutchik cone (see Fig. 10).

This visualization panel can help us in the development of vector summing or norming: it may reveal the possible shortcomings of the computing of the resulting vector (e.g. frequent vector activity in the high intensity area of cone, overly exaggerated movement).

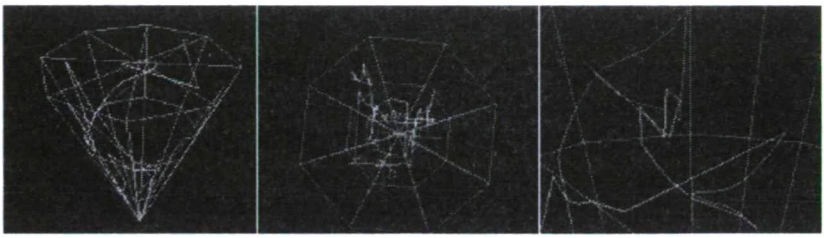


Figure 10: Emotion tracing panel.

7 Testing the GALA Modeler

7.1 Original Transitions

First of all, we have to emphasize that the Plutchik model itself allowed easy transitions in the following cases:

- When the intensity of emotion has changed, it could move into a similar emotional state with increased effect (e.g. from joy to ecstasy)
- An emotion could easily change its immediate neighbor. It was Plutchik's original intention as well to place the similar emotions next to each other. The movement on the surface of the cone realizes this transition.
- When an emotion has a lighter intensity, it could easily switch into another lighter emotion, since their geometrical distance is small (e.g. acceptance to boredom). On the other hand transitions between states that are far away geometrically are less likely (e.g. admiration to boredom).
- A similar category is the switch between very strong emotions. According to the model, a change between strong emotions is less likely than between light emotions. This is also a result of the geometry.
- For the sake of accuracy it is important to note, that in Plutchik's model emotions, moods and cognitive states are mixed together a little (e.g. boredom is more a cognitive state than an emotion etc.). But since we are building an approximate, synthetic model, we can afford this more unified and "standardized" approach. Also, Plutchik's own view strengthens our position as he argues for the unified treatment with good reason. For instance, because of the fact that the difference can be captured by a time component which is handled by the ADSR functions in our extension of the model [12, 23].

7.2 Completely Adequate Behaviors

- Most of the basic observations related to the original Plutchik model were relevant and worked well. Including the change of the length of the emotional

vector (intensity) that resulted in realistic emotional status transitions (rage-anger-annoyance etc.).

- Effects similar to each other, resulted in a stronger effect; the contradictory effects (pensiveness - serenity) weakened or neutralized each other.

7.3 Vector Calculation Issues

Although the summing operation of the vectors for obtaining a resultant vector seems to be straightforward and a computationally economical solution, it does not at all provide appropriate behavior self-explanatory. So, we have to ask ourselves, if it is a good idea to "sum up the emotions" into one resultant emotion?

Of course, in everyday life we cannot talk about the "resultant vector of our emotions". However, when the time comes for a reply, humans also act on the basis of a current dominant emotion and they express that with small variations. Therefore the idea of selecting a dominant emotion is not so far from real-life. Either way, we were anyway forced to select an explicit final emotion since we had to use this for choosing the appropriate animation and search for a relevant text reply.

The dominant emotion selection method can be based on a "weighted summing operation", which is present in humans in a special, unconscious form. It adds up all emotions that happened over time and provide a store, or imprint. That emotional memory is query-able, and one can retrieve the dominant emotional state for example by asking - how are you? Thus the synthetic modeling process may approach this calculation problem similarly.

In several cases the sum operation of some emotional effects do not deliver the expected emotional result status (e.g. $\underline{\mathbf{r}}_{pensiveness} + \underline{\mathbf{r}}_{annoyance} =? \underline{\mathbf{r}}_{boredom}$, which is not really what one would expect from a lifelike model). This issue was clearly foreseeable and was caused by the geometrical nature of the emotional space.

7.4 Issues with Setting the ADSR Functions

It is not easy to pre-program the behavior of the ADSR functions. If the sustain duration of the emotions is too long, there will be too many emotional components present in a certain time that may cause inadequate resulting emotion vector, increasing the distortion of the model. On the other hand, if the sustain interval is too short, the effect of the emotion might improperly be insignificant.

The difference between the two cases can be observed on the two Emotion History panels below (Fig. 11.): the second has much more active components in the same time.

There are possibilities to resolve this conflict. One improvement will be the introduction of a relation matrix, where emotions could inhibit or stimulate each other. This way if a new emotion occurs, it could block the effect of long-term, previously arisen emotions. There is an analogue of this effect in real life. It is well known that, the most recently experienced emotion often is a greater determining factor than its original intensity.

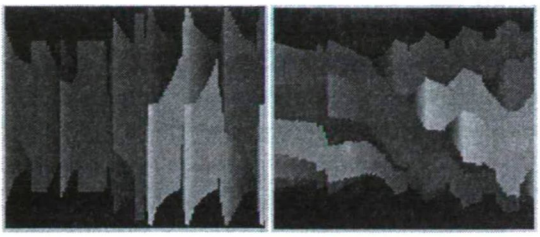


Figure 11: The results of processing the same text with different ADSR settings (different sustain periods).

8 Conclusions and Future Work

We described our attempt to create a chatterbot capable of giving emotionally adequate responses to user utterances. As a starting point we created an architecture for the dialogues system, capable of tracking the current phase of the discussion by analyzing the incoming messages syntactically and semantically, and identifying their role in the dialogue.

Than we created an emotional module to synthesize the incoming user sentences and to generate an appropriate answer to them. We took R. Plutchik theoretical model of emotions and applied it to detect the emotional aspects of natural language communication. We had to make certain amendments to the model itself and ended up developing not "just" a system architecture, but a theoretical model of how communication affects emotional states and how do these states influence communication.

For the proper functioning of our ECA's emotional module, we had to make applications that allow emotional labeling of expressions and sentences and adjusting different attributes (like ADSR data) of them. This also raised interesting theoretical questions, which we tried to test in everyday life and find an answer.

Despite the many questions raised, our emotional model proved to be very practical, useful and a class by itself. It is uniquely synthesizing, mapping and generating emotional responses, decoding the emotional load from sentences and modeling the emotional aspect of the conversation and responding accordingly. We hope these solutions we found are useful for other researchers dealing with dialogue systems, natural language processing, and affective computing.

References

[1] E. André, M. Klesen, P. Gebhard, S. Allen and T. Rist, Integrating models of personality and emotions into lifelike characters, in: A.M. Paiva ed., Affective interactions: Towards a new generation of computer interfaces, Lecture Notes in Artificial Intelligence (Volume 1814) (Springer-Verlag, Berlin, Germany, 2000) 150-165.

[2] J. Bates, The role of emotions in believable agents, Technical report, CMU-CS-94-136, Carnegie Mellon University, 1994.

[3] D. Benyon and K. Höök, Navigation in information spaces: Supporting the individual, in: Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, 1997) 39-46.

[4] A. Damasio, Descartes' error: Emotion, reason, and the human brain (G. P. Putnam's Sons, New York, USA, 1994.

[5] C. Darwin, The expression of emotions in man and animals (D. Appleton and Company, New York, 1872).

[6] D. Dennett, What is thinking, Think Online (2001).

[7] P. Ekman, An argument for basic emotions, in: N. Stein and K. Oatley eds., Basic emotions, (Lawrence Erlbaum, Hove, UK, 1992) 169-200.

[8] M.A. Gilbert, Language, words and expressive speech acts, in: F. van Eemeren et al. eds., Proceedings of the Fourth International Conference of the International Society for the Study of Argumentation, 1999) 231-234.

[9] M.A. Gilbert, Prolegomenon to a pragmatics of emotion, in: Proceedings of the Ontario Society for the Study of Argumentation, (Brock University, St Catherine's, Ontario, Canada, 1997).

[10] D. Goleman, Emotional intelligence (Bantam Books, 1997).

[11] M. Minsky, The society of mind (Voyageur Pub, 1987).

[12] K. Oatley and J.M. Jenkins, Understanding emotions (Blackwell Publishers, Inc., Oxford, UK, 1996).

[13] Országos Széchényi Könyvtár Tezaurusz:
`http://www.oszk.hu/ujdonsag/tezauruj.html`

[14] A. Ortony, On making believable emotional agents believable, in: R.P. Trapple ed., Emotions in humans and artifacts, (MIT Press, Cambridge, USA, 2003).

[15] A. Ortony, G.L. Clore and A. Collins, The cognitive structure of emotions (Cambridge University Press, Cambridge, England, 1988).

[16] R.W. Picard, Affective computing (The MIT Press, Cambridge, Massachusetts, USA, 1997).

[17] P. Rizzo, Why should agents be emotional for entertaining users? A critical analysis, in: A.M. Paiva ed., Affective interactions: Towards a new generation of computer interfaces, Lecture Notes in Artificial Intelligence (Volume 1814) (Springer-Verlag, Berlin, Germany, 2000) 166-181.

[18] H. Schlosberg, Three dimensions of emotions, Psychological Review 61 (1954) 81-88.

[19] S.S. Tomkins, Affect, imagery, consciousness: The negative affects (Springer, New York, 1963).

[20] S.S. Tomkins, Affect, imagery, consciousness: The positive affects (Springer, New York, 1962).

[21] A. Paiva, Affective interactions: Toward a new generation of computer interfaces (Springer-Verlag, 2000).

[22] R. Plutchik, A general psychoevolutionary theory of emotion, in: R. Plutchik and H. Kellerman eds., Emotion theory, research, and experience, (Academic Press, 1980) 3-33.

[23] R. Plutchik, What is mood for? A critique, Psychology 3 (1992).

Received May, 2004

Acta Cybernetica 16 (2004) 643–657.

Telephone Speech Recognition via the Combination of Knowledge Sources in a Segmental Speech Model*

László Tóth† András Kocsor* and Gábor Gosztolya*

Abstract

The currently dominant speech recognition methodology, Hidden Markov Modeling, treats speech as a stochastic random process with very simple mathematical properties. The simplistic assumptions of the model, and especially that of the independence of the observation vectors have been criticized by many in the literature, and alternative solutions have been proposed. One such alternative is segmental modeling, and the OASIS recognizer we have been working on in the recent years belongs to this category. In this paper we go one step further and suggest that we should consider speech recognition as a knowledge source combination problem. We offer a generalized algorithmic framework for this approach and show that both hidden Markov and segmental modeling are a special case of this decoding scheme. In the second part of the paper we describe the current components of the OASIS system and evaluate its performance on a very difficult recognition task, the phonetically balanced sentences of the MTBA Hungarian Telephone Speech Database. Our results show that OASIS outperforms a traditional HMM system in phoneme classification and achieves practically the same recognition scores at the sentence level.

1 Introduction

Although speech recognition requires the fusion of several information sources, it is rarely viewed as an expert combination problem. Such approaches were abandoned in favor of the Hidden Markov Modeling technique (HMM) [13], which treats speech as a stochastic process. The source of the success of HMM is that it offers a sound mathematical framework along with efficient training and evaluation. The price paid for this, however, is that the simplistic mathematical assumptions of the model do not accord with the real behavior of speech. One of these assumptions is the conditional independence of the spectral vectors. Several alternative

*Presented at the 1st Conference on Hungarian Computational Linguistics, December 10–11, 2003, Szeged.

†Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged, H-6720 Szeged, Aradi vértanúk tere 1., Hungary, e-mail: tothl@inf.u-szeged.hu, kocsor@inf.u-szeged.hu, ggabor@inf.u-szeged.hu

models have been proposed, and one of these is the so-called segmental modeling approach [21]. Segmental models treat speech phonemes as one unit – instead of building them up from frames – and thus alleviate the flaw caused by the independence assumption. In recent years our team has been developing a segmental recognition system called OASIS. Our main observation so far is that, although the segmental representation indeed results in a somewhat better phoneme classification performance, these kind of recognizers also have similar robustness problems that traditional HMM systems have. A promising way for obtaining robust recognizers might be to treat speech recognition as a knowledge source combination problem. In this paper we give a generalized speech decoding algorithm created in this spirit. Moreover, we show that both HMM systems and segmental models can be viewed as a special case of this framework. In the second part of the paper we present the current components of the OASIS system, and then perform recognition experiments on the MTBA Hungarian Telephone Speech Database. Knowing that this database contains phonetically balanced sentences recorded from telephone calls from all parts of the country and from people of varying gender and age, this will be quite a difficult recognition task. The performance of the OASIS system will be compared to HTK, known as a sort of standard HMM recognizer in the speech community.

2 Speech Recognition as a Knowledge Source Combination Problem

Although speech recognition is obviously a pattern classification task, the most successful solution, Hidden Markov Modeling, is not a classification algorithm in the strict sense, but a generative model for stochastic random processes. This is because speech recognition does not fit the usual pattern classification framework. That is, most classification algorithms assume that the items to be classified are always represented by the same number of features. In addition, both the dimension of the feature space and the number of classes must be reasonably small. In contrast, speech is a continuous stream of acoustic information. Even if we assume that the talker must stop sometimes, the possible utterances vary in length and their number is practically unlimited. A possible solution is to trace the problem back to the recognition of some properly chosen building blocks. During recognition these building blocks have to be found, identified, and the information they provide needs to be combined. This approach turns speech recognition into a task of classifier combination integrated in a search process.

In the following we present a general speech decoding scheme in the spirit of classifier combination. Firstly, it makes it possible to experiment with alternative combination schemes which could not easily be done within the a traditional HMM framework. Secondly, it provides a more intuitive picture of how the whole recognition process actually works.

Algorithm 1 shows the pseudocode of our generalized speech decoder. Expressed simply, the algorithm works in the following way. Let us assume that our building

Algorithm 1 A Generalized Speech Decoding Algorithm

```
solutions := ∅
hypothesis cue := h_0(t_0, "", 0)
// a hypothesis consists of a time index, a phoneme string, and a score
while there is an extendible hypothesis do
  select an extendible hypothesis H(t, F, w) according to some strategy
  if t = T then
    if only the first solution is required then
      return H
    else
      put H on the list of solutions
    end if
  end if
  for t' = t + 1, t + 2, ··· do
    for all f ∈ F do
      w_f := g_1(f, < t, t' >) // where g_1 estimates the cost of fitting f to < t, t' >
                               // based on the relevant a_i measurements
      w' := g_2(w, w_f) // where g_2 is a proper aggregation function
      if pruning-criterion(w_f, w') then
        construct a new hypothesis H'(t', Ff, w') and put it in the hyp. cue
      end if
    end for
    if stopping-criterion(< t, t' >) then
      break
    end if
  end for
end while
```

blocks are denoted by the elements of the symbol set $\mathcal{F}$. Let the speech signal be given by the series of measurements $A = a_1, \ldots, a_T$. The goal of recognition is to map the speech signal A into a series of symbols $F = f_1 \ldots f_n$, where $f_j \in \mathcal{F}$. The algorithm works from left to right, and stores its partial results in a priority cue. Having processed the signal up to a certain point t, the algorithm looks ahead in time and, from the corresponding measurements, it collects evidence that the next symbol belongs to the time interval being inspected. As neither the exact length nor the identity of the next segment is known, we examine every time index $t' = t + 1, t + 2, \ldots$ that might be the end point of the segment. Each element f of the symbol set is matched to the interval $< t, t' >$, and from each (t', f) pair a new hypothesis is formed and put in the hypothesis cue. As every hypothesis has several extensions, this means creating a search tree. By adjusting the hypothesis selection strategy, the pruning and the stopping criteria one can control how the search space is traversed and pruned.

When the whole signal has been processed, the best scoring leaf is returned as the output result. The score of a hypothesis is calculated in two steps. First, there is a function (g_1) to combine the evidence for each symbol that was collected from the local information sources. Second, this local evidence is combined (via g_2) with the prefix of the hypothesis to obtain a global score. Thus, in effect, classifier combination occurs at two levels.

Obviously, we can have quite different decoders, depending on how the measurements a_i, the symbol set $\mathcal{F}$ and the functions g_1 and g_2 are chosen. Researchers agree only in that g_1 and g_2 should work on probabilistic grounds. In this case Bayes' decision theorem guarantees optimal performance, and statistical pattern recognition provides methods for approximating the probabilities from training corpora.

As regards the selection of the building units, the most reasonable choice is the phoneme since phonemes are the smallest information carrying units of speech (in the sense that the insertion/deletion/substitution of a phoneme can turn a word into another one). Furthermore, in many languages there is an almost one-to-one correspondence between phonemes and letters, so working with phonemes is an obvious choice when converting sound to written text. Nevertheless, smaller or larger units could be used as well. For example, there are arguments that syllables give a more suitable representation of (the English) language. Going the other way, current recognizers mostly decompose phonemes into three articulation phases [13].

The acoustic information sources a_i display the greatest variation from system to system. Traditionally the acoustic signal A is processed in small uniform-sized (20-50 ms) chunks called "frames", and the spectral representation of these serves as direct input for the model. It has been observed, however, that better results are obtained if this representation is augmented with features of longer time-spans so the feature vectors in current systems are a combination of the local and the neighboring 5-50 frames [13].

3 A Special Case: Hidden Markov Models

In spite of its unusual appearance, *Algorithm 1* is not so different from the standard technologies. In particular, its components can be chosen so that it becomes mathematically equivalent to the left-to-right Hidden Markov Models preferred in large-vocabulary speech recognition. In this setup the set of states of the Markov model will play the role of the symbol set in our algorithm. Although the states might simply represent the phonemes of the language, better results are normally obtained if the phonemes are decomposed into three states – one corresponding to the middle steady-state part, and two others describing the transitional phase before and after.

Instead of modeling the class posteriors $P(F|A)$ directly, in speech recognition the product $P(A|F)P(F)$ is normally modelled instead, which leads to the same result but allows one to separate the priors $P(F)$. Building words from states and assessing their prior probability is the problem of language modeling. Here

we assume that $P(F)$ is readily given, and deal only with the acoustic component $P(A|F)$. This factor will be estimated by HMM in the way described below[1].

During processing the HMM goes through a sequence of state transitions. This determines a segmentation based on how long the model stayed in a given state. The probability associated with a given segment sequence is calculated as follows. The probability corresponding to a given segment $S_i =< t, t' >$ and state f is calculated as

$$P(< t, t' > |f) = l_f^{(t'-t)} \cdot \prod_{i=t}^{t'} P(a_i|f), \tag{1}$$

where l_f is a constant between 0 and 1.

The probability corresponding to the whole segment sequence is obtained by multiplying the segmental probabilities:

$$P(A, S|F) = \prod_{i=1}^{n} P(S_i|f_i). \tag{2}$$

In terms of our model, Eq. (1) corresponds to g_1 while Eq. (2) corresponds to g_2. This means that g_2 is simply a multiplication, while g_1 consists of two factors. The term $l_f^{(t'-t)}$ is an exponentially decaying duration model. The product $\prod_{i=t}^{t'} P(a_i|f)$ is a spectral factor that renders a state-conditional likelihood for each measurement of the segment, and then combines these by multiplication – that is, by applying the naive Bayes assumption.

4 An Alternative Technology: Segmental Models

The contradiction between the model that assumes independence and the feature extraction method that makes it patently false has been understood and criticized by many authors [12, 21]. Several cures were suggested, some of them only patching the original HMM algorithm, while some totally abandoning it. The family of segmental models [21] recommends modeling phonemes 'in one', instead of estimating their probabilities by multiplying the frame-based scores. In our framework this means that g_1 (see Eq. (1)) is replaced by some more sophisticated approximation[2]. There are several possibilities to parametrize phonetic segments as one unit. The most popular approach is to create special models that fit parametric curves on the feature trajectories [6, 8, 11, 21]. However, it is also an option to convert the variable-length segmental data into a fixed number of segmental features. What makes this latter method tempting is that this way all the standard classification

[1] Note that we slightly deviate from the standard decomposition into language and acoustic models as, in our notation, the state transitions between the states of a multi-state acoustic model are also included in the language factor, while only the self-transitions of a state are included in the acoustic model.

[2] In contrast to g_1, combination by multiplication at the g_2 level seems quite reasonable because the presence of all phonemes is required for the identity of a word. This makes an AND-like combination logical.

algorithms become applicable to the phoneme classification task. Thus, while the segmental trajectory models are usually built on Gaussian curves, representation by segmental features allows the use of almost any machine learning algorithm. This is why we prefer this approach. In our studies we have reported experiments with a broad range of classifier methods, some of them being very new and not really known by the speech community [17]. Moreover, these classifiers allow the application of such linear and non-linear feature space transformation methods that are currently in the focus of machine learning research. We have published several papers that apply these groundbreaking techniques to phoneme classification [18]. The basic acoustic feature set we invented to represent phonetic segments is very similar to those used in the MIT SUMMIT system [7], but we have added several further features [17]. We have seen similar solutions from other authors, too [3].

A drawback of segmental systems is that the models trained to classify phonetic segments are not necessarily able to handle non-phonetic intervals which is required to find the proper segmentation of a signal. This problem was realized relatively lately [27]. One possible solution is to combine the segmental scores with a frame-based one that assesses the probability belonging to the given segmentation [27]. Another approach is to create artificial "anti-phone" examples and train a classifier to recognize these. We apply discriminative models (neural nets) for this goal, and presented the mathematical formulation of their application in [22]. Later we realized that our solution is similar to that employed in the SUMMIT system [7], but it is build on generative models (Gaussian mixtures) that requires a different formulation. Recently we have proposed another possible solution for the modeling of anti-phone segments by means of replicator neural nets [23].

5 Components of the OASIS Recognition System

The general decoding algorithm of Section 2 forms the core of the OASIS recognition system developed at our institute. That is, the decoding scheme of Algorithm 1 is performed by the 'Matching Engine' component of the system. In the following subsections we will describe in detail what the specific knowledge source components of the system are, how they work, and how they get integrated. As we shall see, the recognition methodology fits the general framework described above, and more actually belongs to the class of segmental models.

5.1 The Phoneme Classifier

The task of the phoneme classifier component is to map a probability to a given $(< t, t' >, f)$ segment-phoneme pair, that is to implement function g_1 of the general decoding scheme. For this we represent each (variable-length) segment by a fixed number of segmental features (for a description of the features see Section 6.2). These segmental features can theoretically be classified by any standard classification method that is able to produce a probabilistic output. Currently the system uses Artificial Neural Nets, but in earlier papers we described our investigations

with many other classifiers as well [17]. Moreover, this component has the option of applying feature transformation algorithms prior to classification. We also wrote several reports on this [18]. In general we find that this modeling scheme results in a 10-30% reduction in the phoneme classification error compared to HMM. This is in accordance with the findings of other authors (see [3] and [11], for example).

5.2 The Anti-Phone Component

During recognition the algorithm will encounter such $< t, t' >$ segments that do not correspond to real phonemes. This may cause two big problems. First, the phoneme classifier might not automatically be able to report these segments. This is the case with our neural network classifier that returns phoneme posteriors and has no output for 'outlier' segments. The second problem is that a manually segmented training corpus contains only examples of real phonemic segments, so these 'anti-phone' segments cannot automatically be trained. One possible solution is to extend the phoneme classifier with an anti-phone class and artificially generate training examples for it [7]. Another option is to assess the probability of a segmentation from frame-based scores [27]. Our system applies the first approach, and the anti-phone probability of a segment is calculated by a complex method, as reported in [22].

5.3 The Language Model

The previous two knowledge sources were acoustic in type. But, of course, in most recognition tasks we have a very serious linguistic restriction on the possible phoneme sequences. The role of the language model is to provide the decoder with the possible phoneme sequences, along with their corresponding probabilities.

When designing the language model component of the OASIS system we initially followed the language description techniques of other recognizers. To be precise, we took the Microsoft Speech API as a starting point. It provides an XML description scheme for the definition of context-free grammars, the words themselves being the terminals of the language. However, in Hungarian listing all the agglutinated forms of a word stem is intractable. As luck would have it, Hungarian morphology can be well modeled by finite state systems. We observed that the agglutinated forms of a stem can be stored in a much smaller space with transducers than with a traditional compression algorithm. This led us to extend the SAPI description so that transducers could be embedded in the place of terminals. This results in a context-free grammar with its terminals being the words recognized by the transducer. Further compression can be achieved by applying special automaton compression algorithms which create the smallest possible transducer that models the same language [15]. Additional savings in storage are possible by storing the resulting transducer with a special data structure [16].

The SAPI handles probabilities by allowing the user to associate weights with the right hand side alternatives of a rule. The transducers embedded in our extended scheme also allow the weighting of the transitions. So, by combining the

two levels, the system is able to associate a probability with any phoneme sequence.

The interface of the language model is adjusted to suit the requirements of the decoding algorithm. During the extension of a hypothesis the algorithm asks for the possible extensions of a phoneme sequence, so the task of the language model is to return all the possible subsequent phonemes of a prefix. Based on this, the interface of the language model consists of two functions, together making it possible to iteratively traverse all the phoneme sequences of the model. These functions are:

`Enter`: Returns the first possible extension of a prefix, along with its probability (or returns a null pointer if there is no extension).

`Next`: Return the next possible extension of the same prefix, along with its probability (or returns a null pointer if there are no more extensions).

As regards the technical details, the implementation of the storage and traversal of the transducers was relatively easy to do. Managing the context-free grammar, however, required the implementation of a stack automaton. We also had to store the actual values of the stack, which led to further technical complications.

5.4 The Combining of the Knowledge Sources

The decoding scheme of algorithm 1 is quite general and thus can be easily extended for the combining of more knowledge sources. An important practical issue is that the more sources we combine the more complex the problem of finding the optimal combination becomes. Fortunately, the problem of knowledge source combination has recently become an active research area. In addition, optimization techniques that support discriminative modelling are getting evermore popular in speech recognition [24]. One such possibility is the Discriminative Model Combination scheme of Beyerlein [1], which optimizes a combination scheme of the form:

$$P(F|A, L_1, \ldots, L_r) \approx \max_S \prod_i P(f_i|A, S)^{\alpha_0} P(f_i|L_1)^{\alpha_1} \cdots P(f_i|L_r)^{\alpha_r}, \qquad (3)$$

where we have r knowledge sources $L_1, \ldots, L_r$ voting on the symbols f_i in the form of posterior probabilities. Combining is then performed by raising the values to a power and multiplying them.

The OASIS system uses Eq. (3) for the combination of the three components. The optimal exponents of Eq. (3) are found by a global optimization algorithm called SNOBFIT [14]. To make the recognition process more efficient we apply multi-stack decoding with several search tree pruning heuristics [9].

6 Experimental Results on the MTBA Database

The goal of this section is to demonstrate the effectiveness of the OASIS system on a real recognition task. For this we chose the phonetically rich sentences of the MTBA Hungarian Telephone Speech Database, because it presents a very general and challenging problem for the acoustic component of any recognizer. Furthermore,

this is currently the largest available speech corpus for Hungarian, and very few results have been reported on it so far. Unfortunately, this recognition task is too general in the sense that there was no way of applying any complex language model. Hence, the tests reported leave the language model component of the system practically unexploited, and assess the performance of the acoustic components only.

6.1 The Corpus

The MTBA Hungarian Telephone Speech Database is the result of an IKTA project carried out in 2001-2003 by the Department of Informatics, University of Szeged, and the Department of Telecommunications and Media Informatics, Technical University of Budapest [28]. Besides other recordings, the database contains 6000 manually labeled and segmented sentences. This part of the corpus was designed so that the phonetic transcript of the sentences contains all possible phoneme pairs that occur in Hungarian. Moreover, the phone callers were organized so that the recordings covered the whole of the country, and the callers were distributed in age and gender. These factors altogether present a very difficult and general recognition task.

For the experiments we first selected those sentences from the database that contained no significant noise and/or half-cut phonemes (denoted by [spk] and [cut] symbols in the phonetic transcript). From the remaining sentences 1367/687 randomly chosen ones were used for training and testing, respectively. These contained 68333/34532 phoneme instances.

6.2 Acoustic Features and Phoneme Classification Scores

For the classification of segments the system applies a 3-layer feed-forward neural net with 200 hidden neurons and a softmax output layer. The net is trained with the minimum cross-entropy training criterion, and training is stopped according to a cross-validation criterion [2]. To find a proper segmental feature set we started from a rather simple representation and gradually extended it with further features. The findings were as follows.

Baseline features. As a traditional frame-based representation, energies in 18 Bark-bands were calculated (via FFT, with triangular weighting and cube root compression) at a frame rate of 333 frames/sec[3]. As the neural net used for segmental classification requires a fixed number of inputs, a conversion is necessary into a fixed-dimensional segmental feature set. At this stage we followed the very simple idea of the SUMMIT system [7]: the band energies were averaged over phoneme thirds, which means a kind of non-uniform smoothing. We may say that the inputs to the neural net are really just average energies in cells that tile the time-frequency space in a special manner.

[3]This is about three times more than the usual 100 frames/sec. We used this increased value because in many experiments we found that it resulted in a slightly better classification performance.

The Importance of Phoneme Duration. In Hungarian most phonemes have a 'short' and a 'long' counterpart, thus duration seems to be a vital piece of information. To model the duration we extended the baseline feature set with a further feature containing the length of the segment. This way the neural net had the opportunity to form any kind of durational description, according to the data. The introduction of the duration feature resulted in a significant error rate reduction, as shown in the table below.

Classification error rate	
Baseline features	Baseline plus duration
47.72%	42.15%

Channel Normalization and Gain Control. The variance in the transfer characteristics of telephone lines is known to have a detrimental effect on speech recognition. A somewhat similar issue is the varying amplitude of the signal. Many normalization techniques have been suggested to counter these effects. Some of them are off-line, which means that they work *after* the whole signal has been recorded (and, consequently, are not suitable for real-time recognition). The on-line algorithms base their processing on the last couple of (centi)seconds. The methods studied do this by means of a 1-pole lowpass filter with time-constant τ.

As the results in the table below show, off-line methods performed slightly better than on-line ones. Out of the on-line methods the non-linear AGC was the best, with a time-constant of 1 second.

Off-line methods	CER%
Mean and dev. normalization (full spectrum)	40.27%
Mean and dev. normalization (per channel)	37.75%
On-line methods	
RASTA filtering	43.86%
Mean and dev. norm. (per channel, $\tau = 250ms$)	41.12%
Mean and dev. norm. (per channel, $\tau = 1sec$)	40.36%
Nonlinear AGC (per channel, $\tau = 250ms$)	39.64%
Nonlinear AGC (per channel, $\tau = 1sec$)	38.49%

Adding Observation Context. In fluent (and fast) speech phones may become so short that they cannot be recognized without their observation context. Auditory research suggests that approximately a 220-250 ms interval contains information about the identity of a phone, but some researchers use observation windows as large as one second [10]. We tried three different settings of the observation length, defined as the phoneme length plus the context length. This means the a variable-sized observation context was considered, depending on the segment size. The context was represented by its average energy values in each Bark-band, thus resulting in two additional feature 'columns' on both sides of the phonemes. As the table shows below, the shortest observation length (150ms) performed best, but this might be due to the large variance of the context over the training set.

	Classification error rate		
Normalization	$\tau = 150msec$	$\tau = 250msec$	$\tau = 1sec$
Off-line mean and dev. norm.	33.18%	34.49%	36.12%
Nonlinear AGC (*1sec*)	33.51%	34.85%	36.25%

Adding Onset and Offset Detectors. Human hearing has cells tuned to detect signal onsets and offsets. These onset and offset detectors may play an important role in the segmentation of a sound stream, especially in finding the boundaries of (certain) phonetic segments. So we implemented an algorithm to simulate these detectors, based on the directions described in [4]. Our detectors calculate the derivatives of the Bark-band energy trajectories and sum them over 3 (6-Bark wide) channels. These curves were evaluated at the phone start and end points and their values were added to the feature set as further features.

We sought to combine these features only with the best feature set found so far. The result shown below indicates that these new features brought only a marginal improvement in the classification scores. We should mention, however, that they proved very important in the phone/antiphone component of the recognition system.

CER with onset/offset feat. (off-line norm., 150ms obs.cont.)	32.17%

6.3 Modeling Anti-Phones

For separating real phonemic segments from the anti-phones, a two-class neural network was used. The segmental feature set was similar to that of the phoneme classifier, but instead of the means of the band energy averages, in this case the *variances* were used. This was a new idea and brought a slight improvement over our previous setup that utilized the same feature set in both the phoneme classifier and the anti-phone model. In all other respects the generation of the anti-phone training examples and their utilization in the decoding process followed the scheme that we presented in [22]. Hence we will refrain from repeating the mathematical formulation here.

Although the anti-phone model could be evaluated in isolation if we generated test examples similar to the generation of the training data, its effect can only really be assessed by its influence on the decoding process. Our results for this will be given in the next subsection.

6.4 Language Model

As the vocabulary of the sentences in the corpus is not restricted in any sense, there was no option of applying any sophisticated (word or morpheme-based) language model. The only thing we could do was to work with a statistical model, e.g. phoneme N-grams. From these we chose the simplest possible one, that is every phoneme was allowed at every position and with the same probability.

6.5 Phoneme Recognition Results

The evaluation of the recognition results is performed by comparing the manual phonetic transcription of a sentence to the transcription generated by the recognizer. Clearly, the recognizer output may contain substitution, insertion and deletion errors as well. To count these the two strings are matched by calculating their edit distance with weights (4,3,3) for substitutions, insertions and deletions, respectively. These weights were proposed by the HTK toolkit [29]. The scores reported below were calculated using the formula

$$Correct = \frac{N - S - D}{N}, \tag{4}$$

where N is the number of all phoneme instances and S and D are the number of substitutions and deletions, respectively. Obviously the recognizer can increase this value by producing many insertion errors as the number of insertions is not accounted for in the formula. To prevent this, the number of insertions was forced to stay around 10-12% by suitably punishing phone transitions in the aggregation formula. This value was suggested by [20].

The table below lists the recognition scores obtained with and without applying the anti-phone model. The figures clearly show the importance of the anti-phone component. Whether these scores are good or not is difficult to judge per se, so in the following section we will furnish some possible bases of comparison.

Sentence-Level Recognition Scores	
Without anti-phones	With anti-phones
53.44%	61.34%

7 Related Work

To our knowledge, apart from us only three teams have used the MTBA corpus so far. Unfortunately, the TSP Lab of the Technical University of Budapest and Hexium Ltd. have performed only isolated word or connected word recognition tests over a restricted vocabulary [5][25]. Although the LSA Lab of the Technical University of Budapest has experimented with a task similar to our setup, in their tests both the train/test division of the data and the phonetic label set were slightly different. Hence, the phoneme recognition score of 55-60% they reported [26] allows only a gross comparison.

To obtain a more precise base for comparison we trained the HTK Toolkit [29], which is a freely available HMM-based recognizer, and is very frequently used to obtain a baseline result when evaluating new technologies. The HTK recognizer was trained with 3-state monophone phoneme models, all having 15 diagonal Gaussian components (this was reported to be about optimal in [5]). Naturally the same train/test setup and phonetic labeling was used as with the OASIS system, and the language model was also set up in a similar way. For signal processing we applied the standard 39-component MFCC vector proposed by the HTK manual.

With these settings HTK recognized 61.60% of the phonemes correctly, with an insertion error rate very close to the one obtained with the OASIS system. This means that our system is capable of practically the same recognition performance as other common recognizers.

Unfortunately, HTK cannot measure phoneme classification directly, so we could not obtain comparative scores to assess the performance of the phoneme classifier module in isolation. However, in an earlier paper we had the option of comparing the phoneme classifier of our system to an HMM-based recognizer for a number recognition task [17]. Moreover, in another paper we tested our phoneme classifier on the TIMIT corpus, for which several classification results are available in the literature [19]. In both cases we found that our segmental representation (along with an ANN or SVM classifier and suitably chosen transformation methods) yields slightly better results than the conventional HMM technology.

8 Discussion and Conclusions

The basic motivation for segmental speech modeling is to replace the simple and incorrect independence assumption of HMM's with a more sophisticated combination scheme. The phoneme classification results indeed show that HMM can be outperformed by even a very simple segmental representation. When it comes to recognition, however, one finds that segmental models need an additional component to handle outlier segments. In our system this problem is handled by the anti-phone models. Our recognition results show that this component can bring up the performance of the system to the level of a traditional HMM recognizer. However, it appears that the gain of better phoneme classification is still lost during decoding, so further developments are required to outperform the current technology. To improve the recognition scores we experiment with alternative anti-phone modeling techniques [23]. Besides this, the addition of further knowledge sources along with the automatic tuning of the parameters in Eq. (3) seems a promising direction. Decomposing the phoneme classifier into many localized experts may improve both performance and robustness, and is a very real trend in speech recognition. Our generalized decoding algorithm provides a good framework for these topics of study, so we plan to investigate them in the near future.

References

[1] P. Beyerlein, Discriminative Model Combination, Proc. ICASSP'98, pp. 481-484., 1998.

[2] C. M. Bishop, Neural Networks for Pattern Recognition, Clarendon Press, 1995.

[3] P. Clarkson and P. J. Moreno, On the Use of Support Vector Machines for Phonetic Classification, Proceedings of ICASSP'99, pp. 585-588, 1999.

[4] E. F. Evans, Modelling Characteristics of Onset-I Cells in Guinea Pig Cochlear Nucleus, Proceedings of the NATO ASI on Computational Hearing, pp. 1-6, 1998.

[5] T. Fegyó, P. Mihajlik and P. Tatai, A Comparative Study on Hungarian Acoustic Model Sets and Training Methods, Proc. Eurospeech 2003, 2003.

[6] H. Gish, K. Ng, A Segmental Speech Model With Applications To Word Spotting, Proceedings of ICASSP'93, pp. 447-450, 1993.

[7] J. R. Glass, A probabilistic framework for feature-based speech recognition, Proceedings of ICSLP'96, pp. 2277-2280, 1996.

[8] Y. Glass, J. P. Haton, Stochastic Trajectory Modeling For Speech Recognition, Proceedings of ICASSP'94, pp. 57-60, 1994.

[9] G. Gosztolya and A. Kocsor, Improving the Multi-stack Decoding Algorithm in a Segment-based Speech Recognizer, Proc. 16th Int. Conf. on IEA/AIE 2003, LNAI 2718, pp. 744-749, Springer Verlag, 2003.

[10] H. Hermansky, Modulation Spectrum In Speech Processing, In: A. Prochazka et al. (eds.), Signal Analysis and Prediction, Birkhauser, pp. 385-398., 1998.

[11] W. J. Holmes, M. J. Russel, Probabilistic-trajectory Segmental HMMs, Computer Speech and Language, V.13, pp. 3-37, 1999.

[12] K. S. Van Horn, A Maximum-entropy Solution to the Frame-dependency Problem in Speech Recognition, Tech. Rep., Dept. of Computer Science, North Dakota State Univ., Nov. 2001.

[13] X. D. Huang, A. Acero and H-W. Hon, Spoken language processing, Prentice Hall, 2001

[14] W. Huyer, A. Neumaier, SNOBFIT - Stable Noisy Optimization by Branch and Fit, Submitted for publication.

[15] A. Kertész-Farkas, Z. Fülöp and A. Kocsor, Magyar nyelvű szótárak tömör reprezentációja nemdeterminisztikus automatákkal, Proc. MSZNY, pp. 231-236, 2003. (in Hungarian)

[16] G. A. Kiraz, Compressed Storage of Sparse Finite-State Transducers, *Proc. of WIA'99* , LNCS Vol. 2214, pp. 109-122, Springer, 2001.

[17] A. Kocsor, L. Tóth, A. Kuba Jr., K. Kovács, M. Jelasity, T. Gyimóthy, J. Csirik, A Comparative Study of Several Feature Space Transformation and Learning Methods for Phoneme Classification, Int. J. Speech Technology, Vol. 3, 3/4, pp. 263-276, 2000.

[18] A. Kocsor and L. Toth, Application of Kernel-Based Feature Space Transformations and Learning Methods to Phoneme Classification, Accepted for Applied Intelligence.

[19] A. Kocsor and L. Toth, Kernel-Based Feature Extraction with a Speech Technology Application, accepted for IEEE Transactions on Signal Processing.

[20] K.-F. Lee and H.-W. Hon, Speaker-Independent Phone Recognition Using Hidden Markov Models, IEEE Trans. ASSP., Vol. 37, No. 11, Nov. 1989.

[21] M. Ostendorf, V. Digalakis and O. A. Kimball, From HMMs to Segment Models: A Unified View of Stochastic Modeling for Speech Recognition, IEEE Trans. ASSP, 4:360–378., 1996.

[22] L. Tóth, A. Kocsor and K. Kovács, A Discriminative Segmental Speech Model and its Application to Hungarian Number Recognition, Proc. TSD'2000, Springer Verlag LNAI Vol. 1902, pp. 307-313, 2000.

[23] L. Tóth and G. Gosztolya, Replicator Neural Network for Outlier Modeling in Segmental Speech Recognition, Proc. ISNN 2004, Springer Verlag LNAI Vol. 3173, pp. 996-1001., 2004.

[24] R. Schlüter, W. Macherey, B. Müller and H. Ney, Comparison of discriminative training criteria and optimization methods for speech recognition, Speech Communication, Vol. 34., pp. 287-310., 2001.

[25] Csaba Szepesvári, personal communication, 2003.

[26] Szabolcs Velkei, personal communication, 2003.

[27] J. Verhasselt et al., Assessing the Importance of the Segmentation Probability in Segment-Based Speech Recognition, Speech Recognition, Vol. 24, (1), pp. 51-72, 1998.

[28] K. Vicsi, L. Toth, A. Kocsor, G. Gordos, J. Csirik, MTBA - magyar nyelvű telefonbeszéd-adatbázis, Híradástechnika, Vol. LVII, No. 8, 2002 (in Hungarian).

[29] S. Young et al., The HMM Toolkit (HTK) (software and manual), `http://htk.eng.cam.ac.uk/`

Received May, 2004

Acta Cybernetica 16 (2004) 659–673.

Phonetic Level Annotation and Segmentation of Hungarian Speech Databases*

Gyula Zsigri† András Kocsor‡ László Tóth‡ and Györgyi Sejtes§

Abstract

The purpose of this paper is to give an outline of phonetic level annotation and segmentation of Hungarian speech databases at the levels of definition and speech technology. In addition to giving guidance to the definition of the content of a database, the technique of annotation and the procedure of manual segmentation, we also discuss mathematical models of computer-aided semi-automatic and automatic segmentation. Finally, we are summing up our observations about the application of the procedures we gained during the processing of the MTBA Hungarian Telephone Speech Database.

1 Designing a Speech Database

Statistics based speech processing, particularly automatic speech recognition, requires well-organized, large speech databases. Training a speech recognition program is based on statistical parameter estimation. Accurate parameter tuning requires training on a large number of samples. A proper training database is made up of collections of such samples accompanied with the necessary notes, labels and transcriptions. The databases should include the observations that are required by the parameter estimation and all the samples that cover the variability of speech (and noises of the environment).

A speech database is a large set of sound data which can be organized by several grouping conditions. The size and internal structure of a database is usually determined by the area of use. To achieve a reliable level of accurate recognition, the material should contain every typical variation that is likely to occur during recognition.

*Presented at the 1st Conference on Hungarian Computational Linguistics, December 10–11, 2003, Szeged.

†Department of Hungarian Linguistics, University of Szeged, H-6722 Szeged, Egyetem utca 2., Hungary, e-mail: zsigri@hung.u-szeged.hu

‡Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged, H-6720 Szeged, Aradi vértanúk tere 1., Hungary, e-mail: kocsor@inf.u-szeged.hu, tothl@inf.u-szeged.hu

§Department of Hungarian Linguistics, University of Szeged, H-6722 Szeged, Egyetem utca 2., Hungary, e-mail: sejtes@hung.u-szeged.hu

1.1 Defining the Content of the Database

The first task is to define the content and standards of the database. According to its purpose, the database may contain sounds uttered individually, words or sentences or spontaneous speech. The second task is to create reading sheets which contain the linguistic units uttered by the informants.

1.2 Quantitative Indices of the Database

The number of necessary records and informants should be determined before starting the recording process. The amount of data required largely depends on the planned application of the speech corpus.

2 Annotation

2.1 The notion of annotation

Annotation means that every sound file is accompanied with a label file which contains descriptor fields related to the sound file's parameters and content [16].

2.2 Criteria for Selecting the Description Information

During the annotation of speech material, informatic, linguistic and social information is attached to every sound record. We bring examples of these accompanying information blocks from the SpeechDat-E [11] corpora and The Budapest Sociolinguistic Interview [9]. SpeechDat is a collection of databases, created by an international expert committee initiated by the European Community. The structure of these databases follows a rule system designed as a guideline for creating databases for training speaker-independent speech recognition programs [16]. This rule system guarantees that the databases created according to the specifications will be similar to each other, therefore they can be used in several speech technology projects.

The Budapest Sociolinguistic Interview (Budapesti szociolingvisztikai interjú) contains the material (with English description) of a sociolinguistic research carried out in the Linguistic Institute of the Hungarian Academy of Sciences.

Based on the above criteria, we suggest using the following variables in annotating databases:

- Informatic: the number of the record, the type of the microphone, subjective level of noise, SAMPA transcript (labeling during recording: beginning, end, obtained data, min, max, stimulus by spelling, labeling by spelling: beginning, end, orthographic record).

- Linguistic: expected lexical material, uttered lexical material, first language and dialect of the informant.

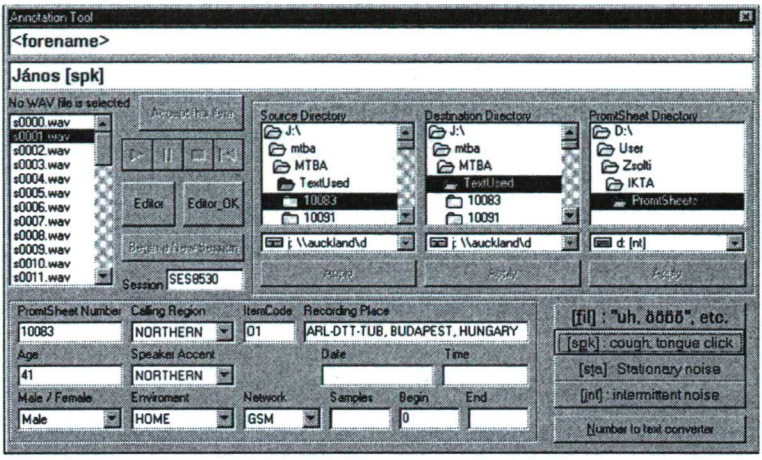


Figure 1: *A possible user interface of the annotation program*

- Social: circumstances (place, date and time of recording), environment, data related to the informant (gender, age, education, place of birth, occupation, state of health, knowledge of foreign languages, speech impediment, time spent abroad).

As an example, Fig. 1 shows the user interface of the software annotation tool developed and used by the Laboratory of Speech Acoustics at the Technical University of Budapest. The descriptor data fields applied in the case of this database can be seen at the bottom-left corner.

3 Segmentation

Many speech databases also contain segmentation information. Segmentation means that, taken as a function of time, the physically observable speech sounds and their boundaries are delimited by start and ending time markers. The purpose of the segmentation is to assign a relation between the speech signal and the phonetic transcript, i.e. which symbol refers to which interval of time. The units of the segmentation are the speech sounds of which phonemes are abstracted [16].

3.1 Methods of Phonetic Level Segmentation and Labeling

The goal of phonetic level segmentation and labeling is to delimit phonetically observable speech sounds and their boundaries as a function of time manually or possibly with the help of an automatic segmentation routine. In this so-called "audio-visual phonetic transcription" of the MTBA database we followed the recommendations of BABEL, an international project dating back to 1997 [17]. The transcription is done by listening to the text and analyzing the time function and/or

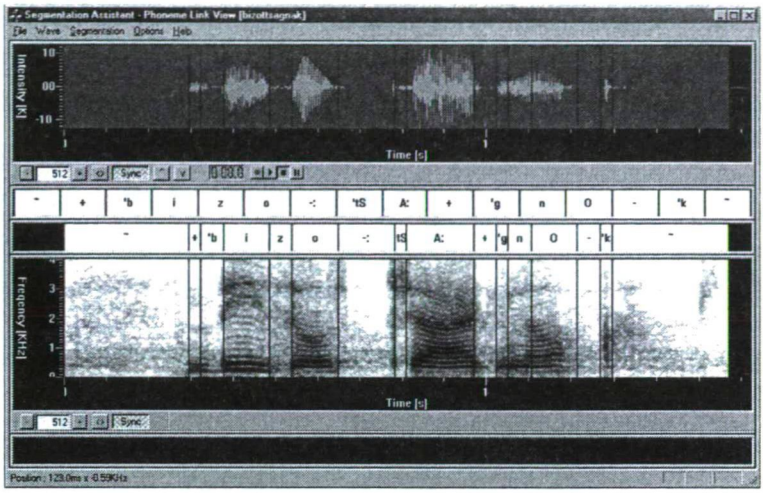


Figure 2: *The user interface of the segmentation program*

the spectrum. For that task, we have developed a special program called the "Segmentation Assistant" (its interface is illustrated in Fig. 2).

The top panel of the program displays the waveform of the speech signal, while the bottom panel shows its spectrogram. In addition to listening, these two visual representations also help the segmentation. The series of phonetic symbols which should be assigned to the given record are displayed in the middle. This phonetic transcript may of course be corrected if the speaker has said something different from what the software has guessed (based on the ortographic transcript). Phonetic symbols and portions of the speech signal are mapped onto each other with boundary markers which automatically align phonetic symbols with sound sections after they have been insterted.

3.1.1 Proposed Labeling Rules

Here we give the labeling rules followed by our team during the processing of the MTBA database:

- SAMPA characters are used for labeling. IPA symbols, widely used by phoneticians and phonologists, are not trivial to map onto computer keyboards. SAMPA, a transcription system developed for computational purposes [1, 18], is convenient in both typing and transmission.
- Characters are positioned between boundary marks.
- Transcription should represent what was actually said.
- Pauses made during an utterance are marked with $\sim$.

- Co-articulatory noises or coughing is marked with {spk}.
- Filled pauses (schwas) are marked with {fil}.
- Noises are marked if they are unambiguously identifiable and do not belong to environmental noises. Transient noise is marked with {int}, stationary noise with {sta}.

3.2 Automatic Methods of Segmentation

Phoneme-level segmentation and labeling is a tedious manual task that requires great care and attention. This work can be made faster and easier by a suitable algorithm that attempts to position the phonetic boundary markers automatically or semi-automatically. Although a perfect algorithmic solution for the phonetic segmentation of a signal is not yet known, even an approximately good positioning of the boundaries can significantly speed up the manual work.

3.2.1 Semi-Automatic Solution for Segmentation

The automatic segmentation of a speech signal into phones is one of the classic problems of speech processing. Currently it is widely accepted that this problem cannot be solved perfectly without at least a partial recognition of the speech signal. That is, methods that are built solely on signal processing techniques and have no machine learning component cannot be expected to provide a perfect solution. Hence we call them "semi-automatic" to emphasize that their output has to be corrected manually.

These methods all operate by measuring the changes in a properly processed version of the signal and assume that the large changes refer to segment boundaries. The main question is, of course, how to process the signal before detecting its changes. This processing should result in a signal that has large jumps at those places where the phonetic quality of the signal changes. Unfortunately, these points do not exactly coincide with large spectral changes. So a more sophisticated processing is required.

The "Segmentation Assistant" software developed by our team performs the following processing. The spectrum is decomposed into four bands. The bands were originally chosen to roughly correspond to formant bands, but we later realized that they practically cover 6 Bark wide ranges on the Bark scale. First we give the formulas that connect the Hertz and the Bark frequency scales:

$$f = 20 + 600 \cdot \sinh(b/6.7),$$

$$b = 6.7 \cdot \operatorname{asinh}\left(\frac{f-20}{600}\right).$$

And the frequency bands processed by the system are:

$$[20Hz; 635Hz],$$
$$[635Hz; 1790Hz],$$
$$[1790Hz; 4490Hz],$$
$$[4490Hz; 11000Hz].$$

The system detects the changes in energy within these f_i bands. The simplest way to measure changes is by examining the derivative. To avoid the detection of minor changes the data is smoothed first. This is performed by the simplest possible method, averaging (however, a more sophisticated filter could obviously be used as well).

$$\hat{f}_i(t) = \frac{1}{2s+1} \sum_{k=-s}^{s} f_i(t+k),$$

where parameter s controls the size of the smoothing window and thus the strength of smoothing.

After this, differentiation is approximated simply by calculating the difference of neighboring (or, depending on parameter d, farther positioned) data values:

$$d_i(t) = |\hat{f}_i(t+d) - \hat{f}_i(t-d)|.$$

The smoothing and differentiating steps described above correspond to a linear system. In contrast to this, it is known that the processing in the human ear has several non-linear steps, too. To simulate these, we also implemented another function that emphasizes the changes in the signal. This function is non-linear and is based on the "adaptive gain control (AGC)" processing characteristic to human hearing. The formula of this function is:

$$y_i(t) = \frac{f_i(t)}{1 + K \cdot \hat{f}_i(t)},$$

where K is a constant that effects the strength of non-linearity.

4. The derivative and the AGC function can be calculated in all four bands and they detect different kinds of spectral changes. In a final step their aggregation is calculated by weighted summation. The largest drawback of the whole method is that the constants occurring in the formulas are all tuned empirically, via tedious experiments. In addition, the result of all three processing steps (smoothing, derivation, AGC) is highly dependent on the spectral resolution (the step size of the analysis window). Currently we have the optimal parameters only for certain special cases.

3.2.2 Automatic Segmentation

The best automatic segmentation algorithms are built on machine learning. Specially, the speech recognition algorithms themselves can also be used for speech

segmentation. This is because during the recognition of a sentence recognizers perform a search: they try to fit every possible phonetic transcript on the underlying signal. Moreover, the recognizers also try every possible segmentation, since not only the transcript but also the segment boundaries are unknown. The result of the recognition consists of the phonetic transcript and the segmentation that was found to give the best fit. In a speech recognition application the segment boundaries are not required, so this component of the result is ignored. However, this "hidden" feature of the recognizer can be exploited for segmentation. In this case only one possible transcript is given to the recognizer, so it practically has to find only the best fitting segmentation for the given transcript. This kind of usage of a recognizer is commonly called "forced alignment" in the literature.

Based on the concept above the OASIS recognizer developed by our team was adjusted to the needs of automatic segmentation. The recognizer itself is segment-based and utilizes and artificial neural network for segmental classification. About its phoneme classifier component more details can be found in [8]; the sentence-level buildup of the system was described in [13]. In the following we shortly present the mathematical background of the system.

The mathematical formulation of speech recognition and automatic segmentation. The speech recognition algorithms are usually based on statistics and probability theory. Their starting point is the decision theoretic theorem that states that to minimize the number of misclassifications one always has to chose the (a posteriori) most probable choice [2]. More specifically, if X denotes the object to be classified and W the possible classes (in our case the phonetic transcripts), then the output $\hat{W}$returned by the recognizer will be:

$$\hat{W} = \operatorname*{argmax}_{W} P(W|X).$$

From this decision rule it follows that the goal of machine learning is to model the distribution $P(W|X)$ as precisely as possible. However, since the number of possible W sentences and the space of the possible X acoustic observations is too large, W and X both have to be decomposed somehow. To put it simply, it is impossible to model whole sentences, so we have to build them from some smaller units. The most reasonable choice for a building block is the phoneme. Let us suppose that the phonetic transcript W is a string of phonetic symbols, that is $W = w_1 w_2 ... w_N$. During recognition not only the identity of the building blocks are unknown, but also their number and position in the signal. Supposing that their acoustic counterpart in the signal can be practically anywhere, the best we can do is to examine every possibility. Let S denote the set of all possible segmentations of observation X. Then, according to the corresponding rule of probability theory

$$P(W|X) = \sum_{S} P(W|X,S) \cdot P(S|X).$$

If we presume that the signal has one "correct" segmentation and all the others are very improbable, then the summation can be approximated by maximization:

$$P(W|X) \approx \max_{S} P(W|X,S) \cdot P(S|X).$$

Embedding this formula in the first one we obtain that (phoneme-based) speech recognition practically means a search along two dimensions. First, we have to find the transcript W that fits X the best; second, for each W we have to find the best segmentation S. From this it obviously follows that a recognizer working by the above principle can easily be used for automatic segmentation as well, as during recognition it implicitly finds the optimal segmentation of the input signal. The only difference is that when used for automatic segmentation search over W can be omitted as the phonetic transcript is readily given.

Speech recognition and segmentation with neural nets. Most speech recognizers used in practice are based on the Hidden Markov Modeling technology [3]. These – exploiting the Bayes theorem – model the $P(X|W)$ so-called "class-conditional" ditribution instead of the $P(W|X)$ posteriors. (but this fact does not significantly influence the previous arguments). Modeling the class-conditionals means that the distribution of the acoustic observations that belong to each phoneme symbol is approximated separately, usually in the form of weighted Gaussian mixtures.

A possible alternative is to apply neural nets instead. These however model the $P(W|X)$ posteriors [2]. As our recognizer is built on neural networks, in the following we detail only this case.

Let us analyze the approximating formula of $P(W|X)$ given earlier. As we see, it consist of the product of two factors. Let us have a closer look at the first one, $P(W|X,S)$. As we already mentioned, it has to be decomposed because of the too large number of the possible W and X values. For this we have to make independency assumptions regarding the distribution $P(W|X,S)$. Our first such assumption will be that the neighboring phonemes occur independently[1]. With this we obtain that

$$P(W|X,S) = \prod_{i=1}^{N} P(w_i|X,S).$$

The other independency assumption is that the quality of a phoneme does not depend on the full acoustic signal but only on that part that belongs to the phoneme, according to the segmentation being evaluated. Let this signal excerpt be denoted by $X_{i(S)}$. Then

$$P(W|X,S) = \prod_{i=1}^{N} P(w_i|X_{i(S)}).$$

That is, distribution $P(w_i|X_{i(S)})$ tells us the probability of a certain phoneme symbol w_i belonging to a given acoustic segment $X_{i(S)}$. This is why this component of the system will be called the phoneme classifier. This probability can be properly modeled by neural nets; the only requirement is that the segments (that are of

[1]This obviously does not hold and usually the language model of the recognizer is responsible for modeling the correlation between the neighboring symbols. In our case, however, it will not be required, as we have only one and readily given phonetic transcript.

varying length in general) should always be represented by the same number of features.

The other component of the formula is $P(S|X)$. Its role is to assign a probability to every possible segmentation. It also has to be decomposed, if we want to learn it by neural nets. For this we again need an independency assumption. We will assume that the segments s_i independently influence the probability of the whole segmentation. Furthermore, as the possible segmentations compete with each other, the formula will contain not only the segments of the currently inspected segmentation but also all the other segments of the other segmentations. Based on this, our approximation will be:

$$P(S|X) = \prod_{s_i \in S} P(s_i|X_{s_i}) \prod_{\bar{s}_i \in \bar{S}} P(\bar{s}_i|X_{\bar{s}_i}),$$

where $s \in S$ denotes the segments of the segmentation under evaluation and $\bar{s} \in \bar{S}$ denotes all the other segments occurring in any other segmentation. $P(s_i|.)$ denotes the probability that a segment is indeed phonetic, while $P(\bar{s}_i|.)$ is the probability that the segment does not correspond to a phone, that is, it is a so-called "anti-phone" (that is, part of a phone or longer than a real phone). This two-class classification of speech segments is again well learnable by neural nets.

In practice the number of possible segments is too large, so we simplified the formula above one step further by not considering all the elements $\bar{s} \in \bar{S}$ but only those that are the closest to s [13].

3.3 Manual Segmentation

Although automatic or semi-automatic segmentation has promising results, really accurate segmentation requires manual work even if it is tedious and time-consuming.

3.3.1 Procedure of Manual Segmentation

The material prepared for segmentation is accompanied with phonetic transcripts. Orthographic records are transformed into phonetic transcripts with respect to the rules of Hungarian pronunciation (e.g. rules of assimilations, etc.). This type of transcription is referred to as phonotypic transcription.

The work begins with cleaning up the recorded material. After the first listening, longer pauses, repetitions and words that are not listed in the vocabulary source are deleted from the wav files. Next, segmentation is done with a proper software tool – in our case we always use HotSA, a program developed by the Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and the University of Szeged. The process of manual segmentation usually consists of the following steps. The material is listened to again to filter the noise, lexical errors or transcription errors that remained after the first listening. Then word boundaries are marked in a zoomed-in spectrogram. During the third listening, boundaries and phonetic symbols are finalized. As a final step, the segmentation information

is saved in a phonetic transcription file in the form of a sequence of phonetic codes, along with starting and ending time information. If required, informants' errors and non-environmental noises may be booked, too.

3.3.2 Criteria for Segmentation

During the manual segmentation of the MTBA database we followed the rules given below. These are again based on the earlier segmentation expertise of our cooperating partner, Laboratory of Speech Acoustics of the Technical University of Budapest [16]. These rules are required because it is idealistic to assume that speech sounds form sound sequences in the linear sound structure. In reality, sounds are not discrete units with clear-cut boundaries between them, but they have shorter or longer overlaps. Nevertheless, boundaries must be inserted. Sometimes neither comparing the spectrogram with the wave form, nor repeated listening gives us objective cues to find the boundaries accurately. In every case we followed these guiding principles:

- Segment boundaries are aligned with null transition.
- With voiced sounds, zero transition refers to positive null transition. The boundary must be marked very precisely. With voiceless sounds, the beginning of the sound can be marked with 1 ms accuracy.
- The beginning of a vowel should be marked at the beginning of voicing (after a voiceless sound).
- The beginning of stops and affricates are marked after the last period of the preceding sound.
- In vowel-vowel or vowel-resonant sequences, the boundary is inserted at 50% of the overlap. This is less accurate because the separation of the sounds is unsure [17].
- The [cut] code is used for sounds that are not identifiable even after repeated listening.

The following procedure has proved to be useful: first we listened to the record, paying attention to noises, speech errors or transcription errors. Then at small resolution (as in Fig. 2), raw segmentation is done. Finally, at large zoom the boundaries and phonetic symbols are finalized.

3.4 Observations

During the manual segmentation of the MTBA corpus we encountered many interesting cases and collected a large set of observations. The most important and general of these was that coarticulation may cause difficulty in the clear perception of the overlap or even either or both sounds. Hence we are planning to examine the overlaps thoroughly to improve the notation.

More specially, we had several findings regarding to certain phones or phone pairs. We list some of these interesting observations below.

The [l] and [j] sounds are difficult to distinguish. They have no clear boundaries in neither their wave forms, nor their spectrograms, even though both sounds are clearly audible. It is likely that the criteria used for consonants are suitable for separating [j] and [l] because both sounds are sonorants like vowels. Furthermore, their pronunciation is strongly influenced by the adjacent vowels (e.g. they become round next to round vowels). There is no clear division in the spectrograms of these consonants and the vowels next to them. Consequently, [l] and [j] should be identified using the criteria for vowels, not the criteria for consonants.

Fricatives such as [s], [z], [S], [Z] are easily identifiable due to their strong noise. The intervocalic allophone of /v/ is difficult to find because it has very little noise. Clusters of nasals such as [mn] have usually clear spectrograms but clusters like [mb] (as in gomba 'mushroom' or bim-bam 'ding-dong') are difficult to segment because the voiced closure is often missing.

Distinguishing voiced and voiceless h is not unambiguous, nevertheless we are trying to separate the allophones whenever possible.

Word-final or syllable-final stops (k, g, t) are often unreleased. Segmentation procedures that localize stops in their release phase, will fail here even though humans succeed by identifying them from the beginning of the closure.

Voiced g is often unreleased.

Long consonants get shortened between long vowels.

3.4.1 The problem of Closures

Speech recognition programs learn to analyze sound waves as sequences of discrete speech sounds by receiving large amount of digitalized sound recordings annotated with manually inserted boundary markers and phonetic transcriptions. Boundary markers are only approximate due to the nature of the speech continuum. A speech sound does not end before the next one begins but there is a considerable overlap between them. Boundary markers are not meant to exactly delimit speech sounds. Their role is to tell the program approximately where to look for information. Vowels and continuant consonants can be relatively well recognized from their innermost portions only. However, the outermost portions (the transitions phases) play a very important role in the perception of stops.

Papp (1996) and many other authors distinguish three phases of a stop:

- closure (implosion): beginning phase
- hold (occlusion): middle phase
- release (explosion): final phase

The hold phase of voiceless stops is completely silent but even voiced stops cannot be identified from the week voicing of their hold phase without place information. If the speech recognition program ignored the closure and release phase of stops, then it would have to identify the stops from sheer silence or voicing.

In speech recognition the easiest way to identify a stop is to examine its release phase. However, it only works if the stop has release. If there is no release then the information can be gained from the closure, which a speech recognizer can take into consideration only if context-dependent models are used. It is interesting that Hungarian textbooks of phonetics do not mention the importance of closure. According to Papp [10], the beginning and the middle phase is normally silent, sound only occurs in the final phase. R. Molnar [12] or Kassai [4, 5, 6] teach the same.

Ignoring the closure is less problematic in Hungarian than in some other languages because most Hungarian stops are released. But not all of them. A stop followed by a homorganic nasal (as in népmese [pm] 'folk tale,' kötni [tn] 'to bind' or satnya [t'N] 'sickly') is only released if there is a pause between the stop and the nasal. Less frequently, stops may be unreleased at the end of the word or in other stop + consonant clusters. Unreleased stops are only ignored by popular university textbooks. Vértes O. [14, 15] mentions them several times.

If satisfied with a speech recognition program which recognizes stop most of the time then we may choose the easier solution and ignore unreleased stops. When using so-called context independent phone models, the closure phase is mapped to the end of a preceding vowel, so the model of the stop ignores it. This is just in accordance with the manual segmentation of the word. Figure 3 shows an example – the segmentation of nap 'sun':

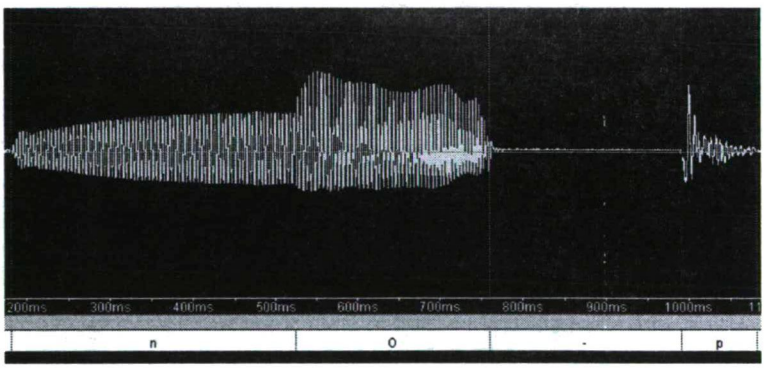


Figure 3: *The segmentation of nap 'sun'*

Note the release of high amplitude after the hold phase marked with minus.

The release of word-final stops is sometimes less powerful, as can be seen in Figure 4.

Listening to what is marked as [O] in nap or [E] in szöveg, humans are able to identify the following stop from the closure phase itself. Simple speech recognizers that apply only context-independent phoneme models ignore this acoustic cue. Even by ignoring it, the program will successfully classify the stop based on its release only.

If we want to model human perception then the representation of a stop should

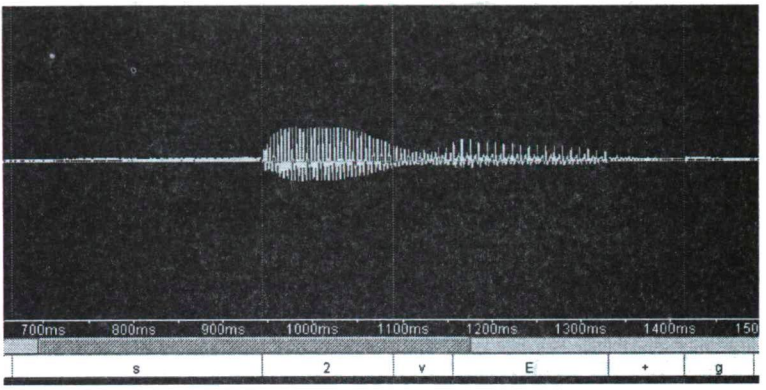


Figure 4: *Release of word-final voiced stop in szöveg 'text'*

include all three phases from closure to release. That is, the model should also examine the closure phase, which corresponds to the segmentation demonstrated in Figure 5.

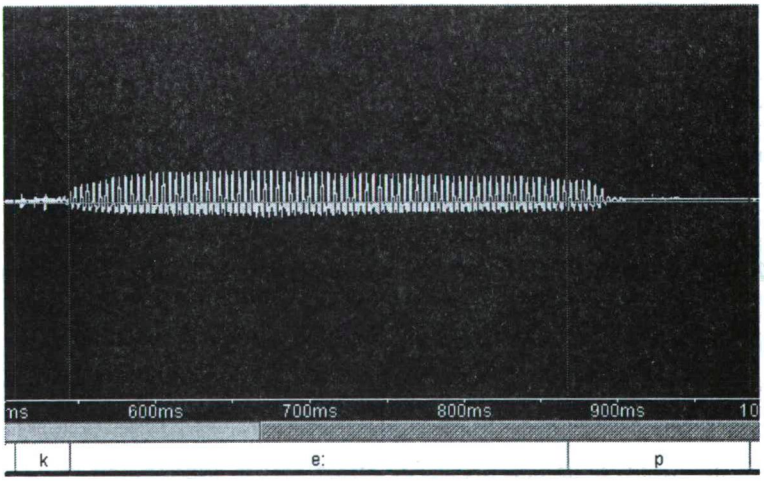


Figure 5: *Segmentation of kép 'picture' with unreleased [p]*

This way the acoustic modelling procedure becomes slightly more complicated and requires context-dependent models, but it makes it possible to recognize both released and unreleased stops.

4 Summary

We have summed up the principles and possibilities of database creation, annotation and segmentation. Special emphasis was given to the fact that such databases may be helpful to speech technology in the development (training and testing) of speaker-independent speech recognition programs. This will open up the road to develop different services of speech technology. The collected sound material is also useful for linguistic or speech technological research thanks to the segmentation and annotation information it contains.

5 Acknowledgments

The creation of the MTBA Hungarian Telephone Speech Corpus was supported by the IKTA grant No. 00055/2001 of the Hungarian Ministry of Education. It was collected and processed in cooperation by the Department of Informatics, University of Szeged and the Department of Telecommunications and Media Informatics, Technical University of Budapest.

References

[1] Barry, W.J. and Fourcin, A. J., Levels of labelling, Computer Speech and Language, Vol. 6 1992, pp. 1–14.

[2] Duda, R. O., Hart, P. E., Stork, D. G., Pattern Classification, Wiley and Sons, 2001.

[3] Huang, X., Acero, A., Hon, H.-W., Spoken Language Processing, Prentice Hall, 2001.

[4] Kassai I., A fonetikai háttér, in Kiefer Ferenc (szerk.) Strukturális magyar nyelvtan 2: Fonológia, Budapest, Akadémiai Kiadó, 1994, pp. 581–665.

[5] Kassai I., Fonetika, Budapest, Nemzeti Tankönyvkiadó, 1998.

[6] Kassai I., Fonetika, in Kiefer Ferenc – Siptár Péter (szerk.) A magyar nyelv kézikönyve, Budapest, Akadémiai Kiadó, 2003, pp. 507–548.

[7] Kiss J., Magyar dialektológia, Budapest, Osiris Kiadó, 2001.

[8] Kocsor, A., Tóth, L., Kuba, A. Jr., Kovács, K., Jelasity, M., Gyimóthy, T., Csirik, J., A Comparative Study of Several Feature Space Transformation and Learning Methods for Phoneme Classification, International Journal of Speech Technology, Vol. 3, Number 3/4, 2000, pp. 263–276.

[9] Kontra, M., Váradi, T., The Budapest Sociolinguistic Interview: Version 3. Linguistics Institute. Budapest, Hungarian Academy of Science, 1997.

[10] Papp I.,Leíró magyar hangtan, Budapest, Tankönyvkiadó, 1966.

[11] Pollak, P., Cernocky, J., Boudy, J., Choukri, K., Heuvel, H., Vicsi, K., Virag, A., Siemund, R., Majewski, W., Sadowski, J., Staroniewicz, P., Tropf, H., Kochanina, J., Ostrouchov, A., Rusko, M., Trnka, M., SpeechDat(E) -Eastern European Telephone Speech Databases Proceeding LREC' Satelite workshop XLDB – Very large Telephone Speech Databases, Athens, 2000.

[12] R. Molnár E., Leíró magyar hangtan, Budapest, Tankönyvkiadó, 1989.

[13] Tóth L., Kocsor A., Kovács, K., A Discriminative Segmental Speech Model and its Application to Hungarian Number Recognition, in: P. Sojka, I kopecek, K. Pala (eds.): TSD'2000, LNAI 1902, Springer Verlag, 2000, pp. 307–313.

[14] Vértes, O. A., Bevezetés a fonetikába. Második, bővített kiadás, Budapest, Gyógypedagógiai Tanárképző Főiskola, 1952.

[15] Vértes, O. A. Az artikuláció akusztikus vetülete, in Bolla Kálmán (szerk.), Fejezetek a magyar leíró hangtanból, 155–164, Budapest: Akadémiai Kiadó, 1982.

[16] Vicsi, K., Tóth, L., Kocsor, A., Gordos, G., Csirik, J., MTBA-Magyar nyelvű telefonbeszéd-adatbázis, Híradástechnika, LVII. 2002/8, Budapest, pp. 35–43.

[17] Vicsi K., Vig A., Az első magyarnyelvű beszédadatbázis, Beszédkutatás '98, MTA Nyelvtudományi Intézete, Budapest 1998, pp. 163–177.

[18] Wells, J. at all., Standard Computer-Compatible Transcription, Esprit Project 2589 (SAM), Doc. no. SAM-UCL-037. London: Phonetics and Linguistics Dept., UCL, 1992.

Received May, 2004

CONTENTS

Preface 507
Björn Borchardt: A Pumping Lemma and Decidability Problems for Recognizable Tree Series 509
Ljubomir Cvetković: An addition to the methods of test determination for fault detection in combinational circuits 545
Zoltán L. Németh: A Hierarchy Theorem for Regular Languages over Free Bisemigroups 567
Nándor Sieben: Hexagonal polyomino weak (1,2)-achievement games 579
Gábor Alberti, Judit Kleiber, and Anita Viszket: GeLexi project: Sentence Parsing Based on a GEnerative LEXIcon 587
János Bácsi: A Semantic Network Dictionary for Dysphasia Therapy 601
András Hócza: Noun Phrase Recognition with Tree Patterns 611
Gábor Tatai and László Laufer: Extraction of Affective Components from Texts 625
László Tóth, András Kocsor, and Gábor Gosztolya: Telephone Speech Recognition via the Combination of Knowledge Sources in a Segmental Speech Model 643
Gyula Zsigri, András Kocsor, László Tóth, and Györgyi Sejtes: Phonetic Level Annotation and Segmentation of Hungarian Speech Databases . . . 659

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János